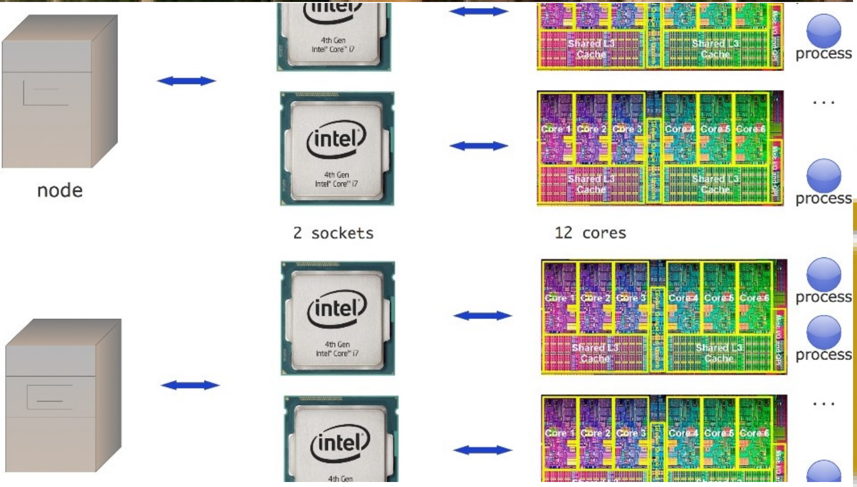
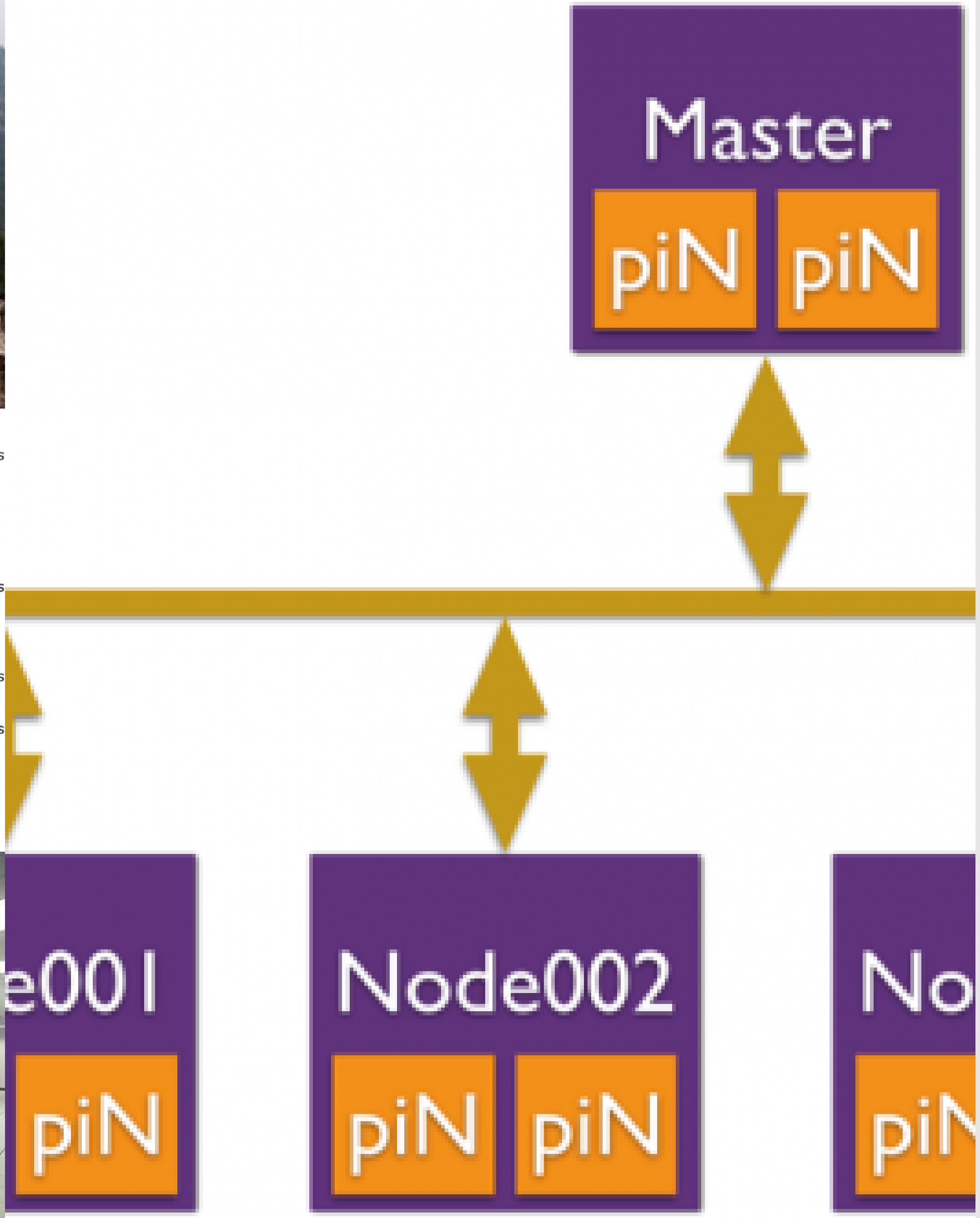
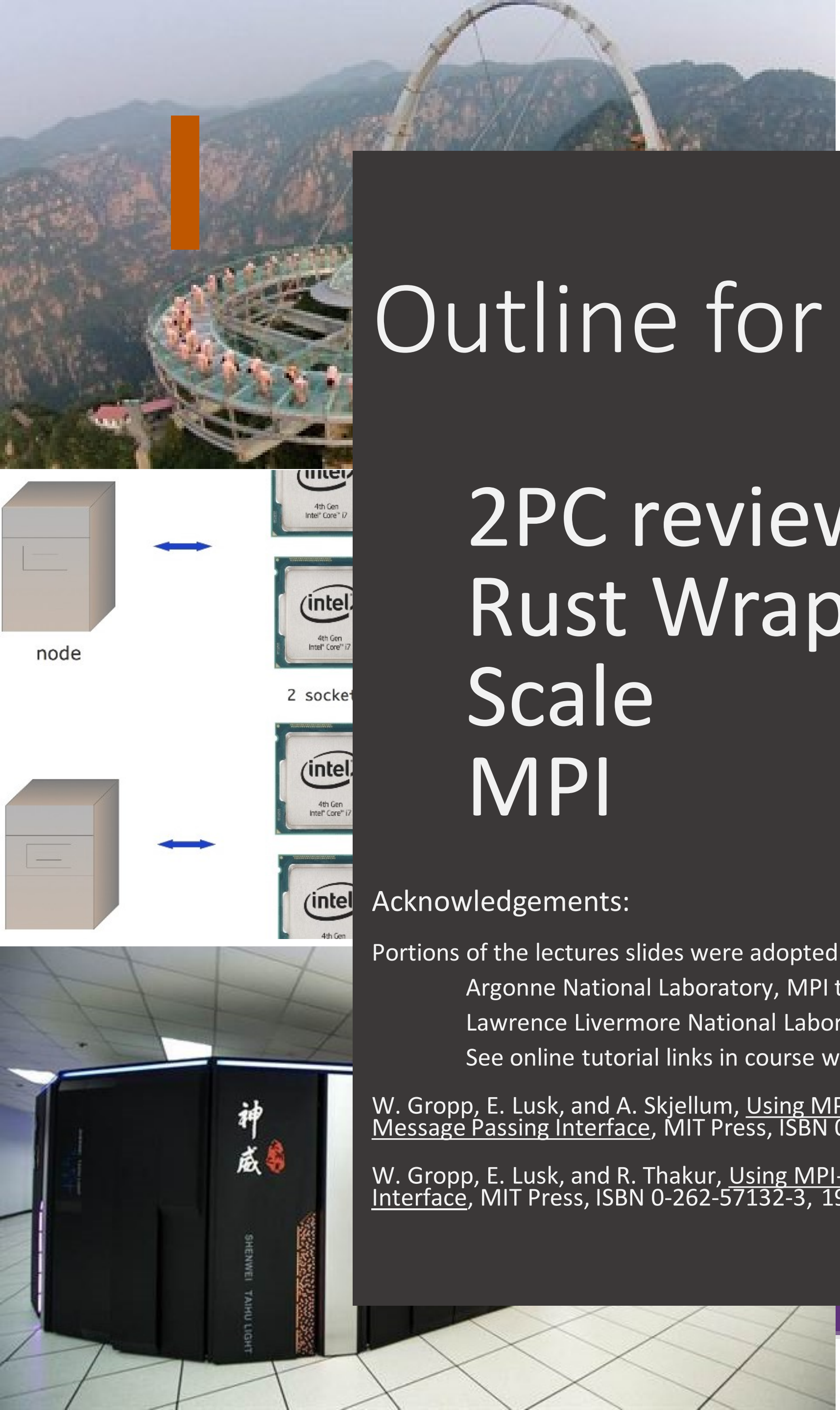
Parallelism at Scale: MPI

cs378h


node
2 sockets
12 cores
process
process
process
process


神威


Master
piN
piN
Node002
piN
piN
piN



Outline for Today

- 2PC review
- Rust Wrapup
- Scale
- MPI

Acknowledgements:

Portions of the lectures slides were adopted from:

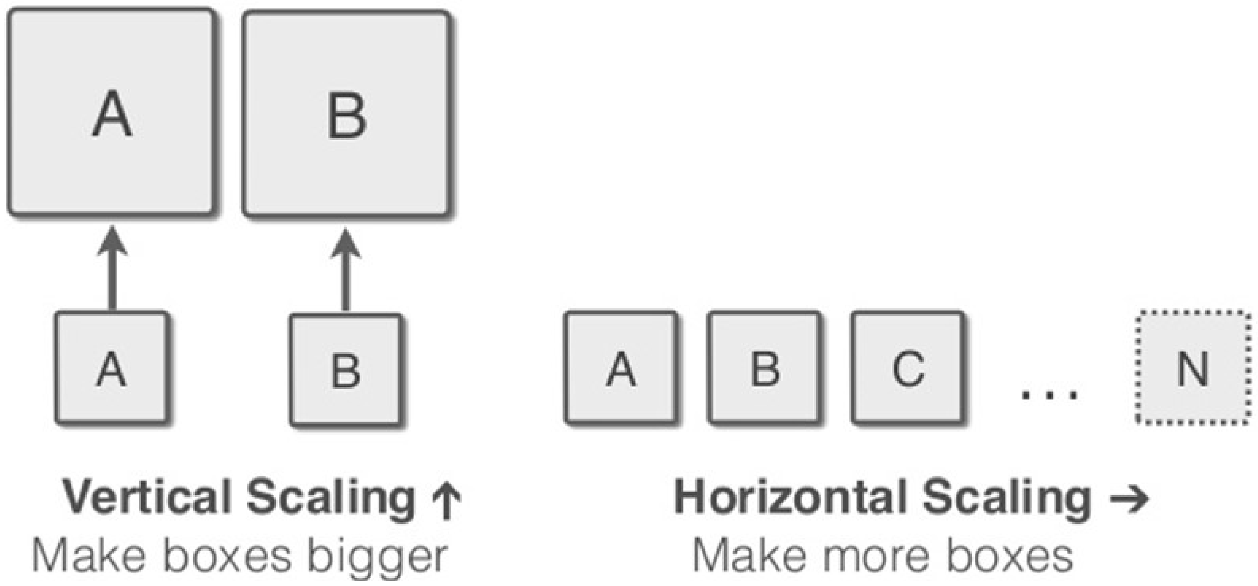
- Argonne National Laboratory, MPI tutorials.
- Lawrence Livermore National Laboratory, MPI tutorials
- See online tutorial links in course webpage

W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press, ISBN 0-262-57133-1, 1999.

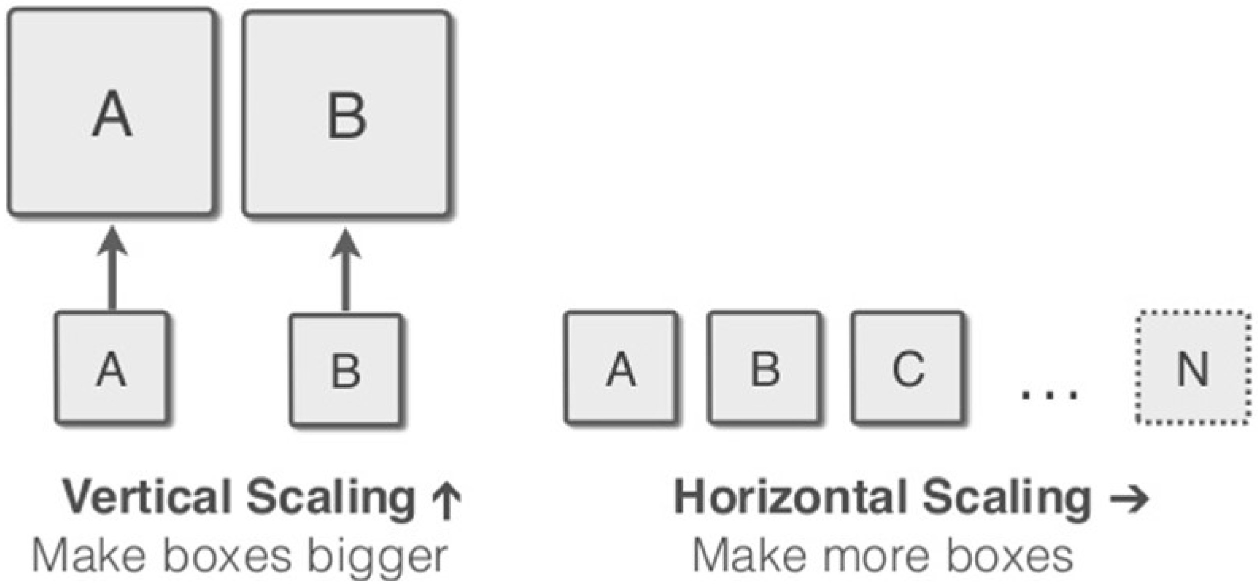
W. Gropp, E. Lusk, and R. Thakur, Using MPI-2: Advanced Features of the Message Passing Interface, MIT Press, ISBN 0-262-57132-3, 1999.

Scale Out vs Scale Up

Scale Out vs Scale Up



Scale Out vs Scale Up



Vertical Scaling	Horizontal Scaling
Higher Capital Investment	On Demand Investment
Utilization concerns	Utilization can be optimized
Relatively Quicker and works with the current design	Relatively more time consuming and needs redesigning
Limiting Scale	Internet Scale

Parallel Systems Architects Wanted

Parallel Systems Architects Wanted

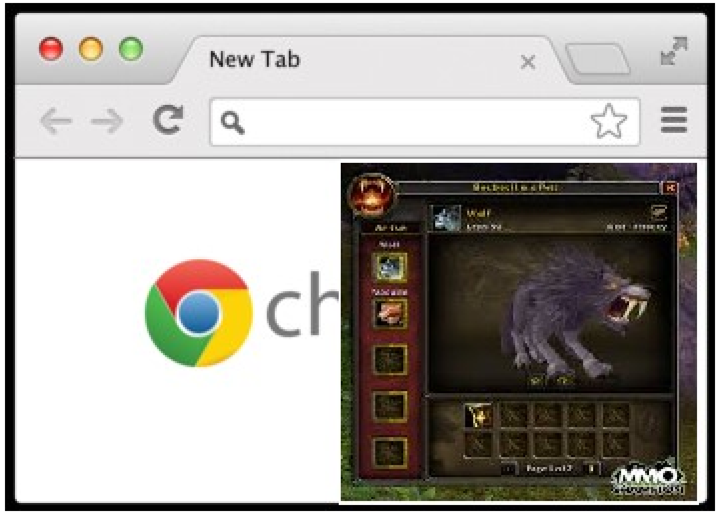
Hot Startup Idea:

www.purchase-a-pooch.biz

Parallel Systems Architects Wanted

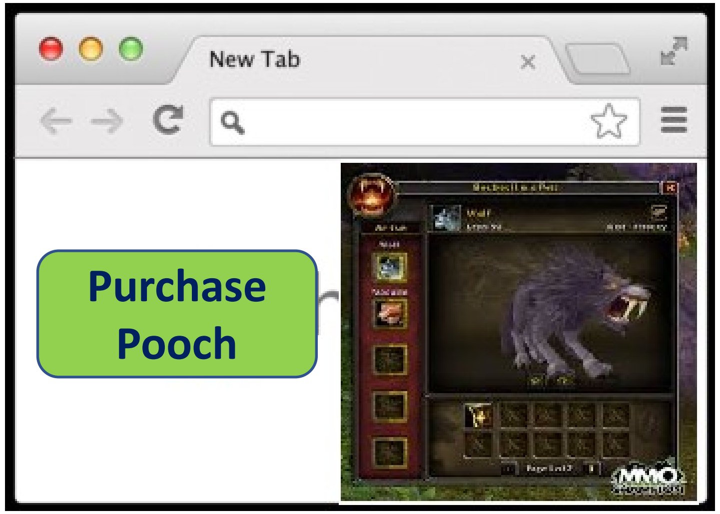
Parallel Systems Architects Wanted

1. User Browses Potential Pets



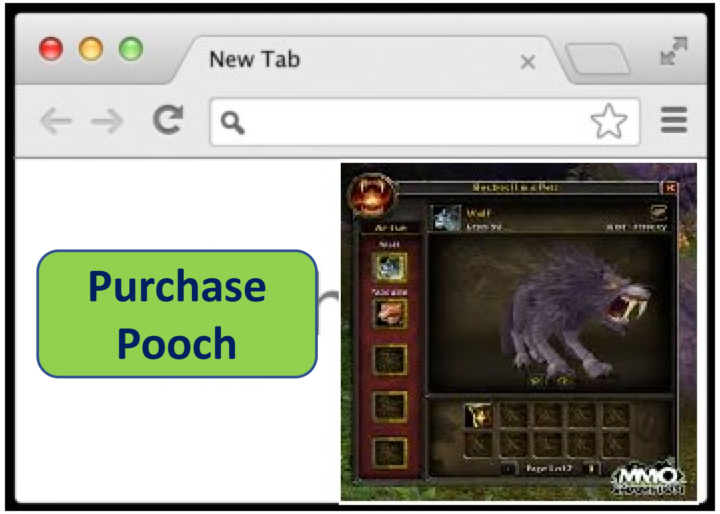
Parallel Systems Architects Wanted

1. User Browses Potential Pets
2. Clicks “Purchase Pooch”



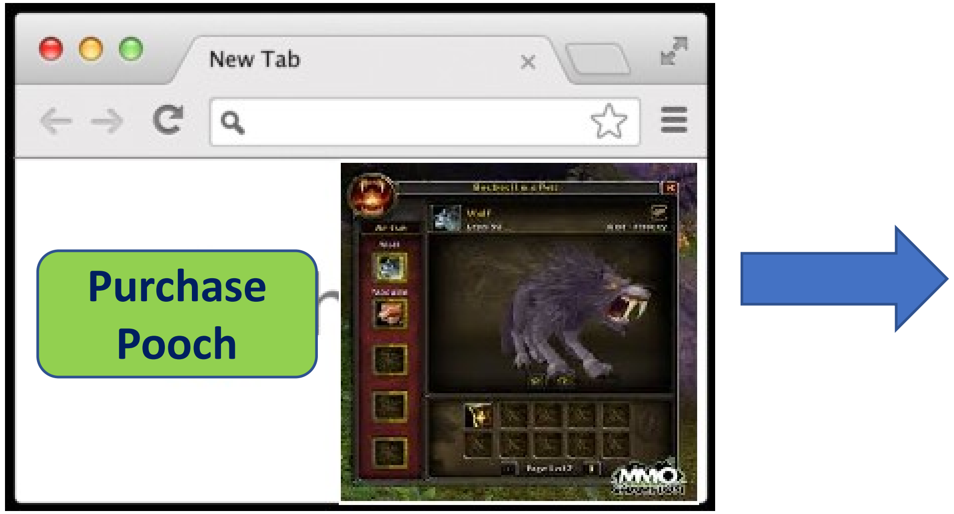
Parallel Systems Architects Wanted

1. User Browses Potential Pets
2. Clicks "Purchase Pooch"
3. Web Server, CGI/EJB + Database complete request



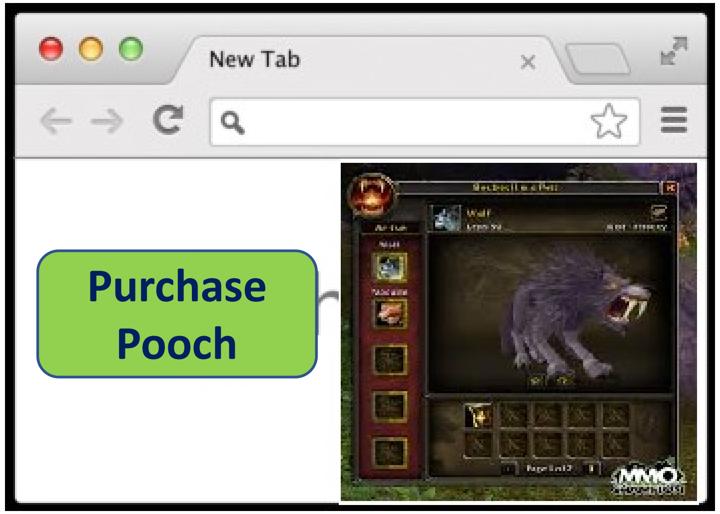
Parallel Systems Architects Wanted

1. User Browses Potential Pets
2. Clicks "Purchase Pooch"
3. Web Server, CGI/EJB + Database complete request
4. Pooch delivered (not shown)



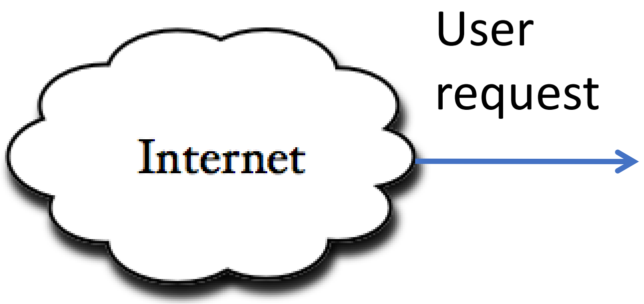
Parallel Systems Architects Wanted

1. User Browses Potential Pets
2. Clicks "Purchase Pooch"
3. Web Server, CGI/EJB + Database complete request
4. Pooch delivered (not shown)

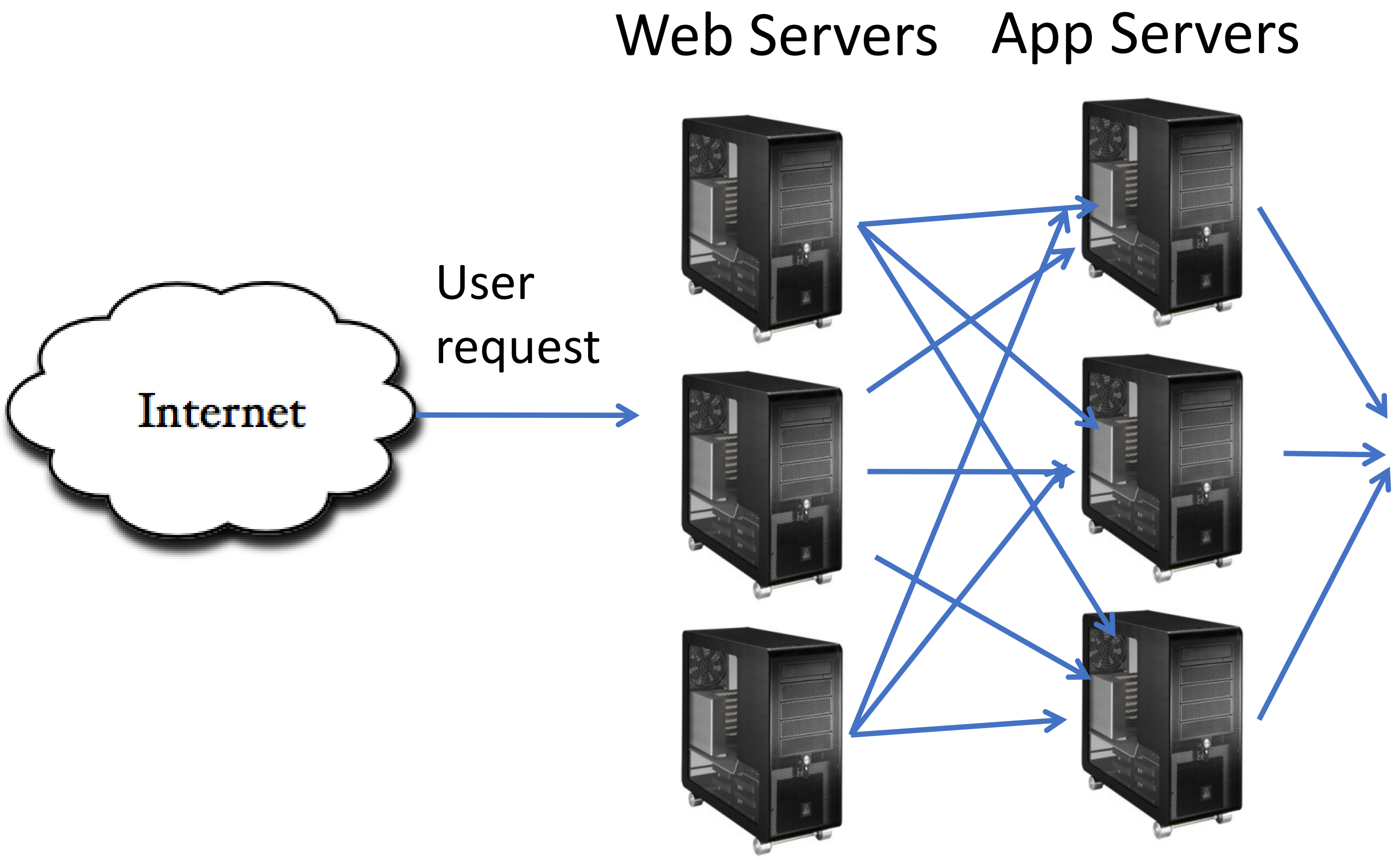


How to handle lots and lots of dogs?

3 Tier architecture

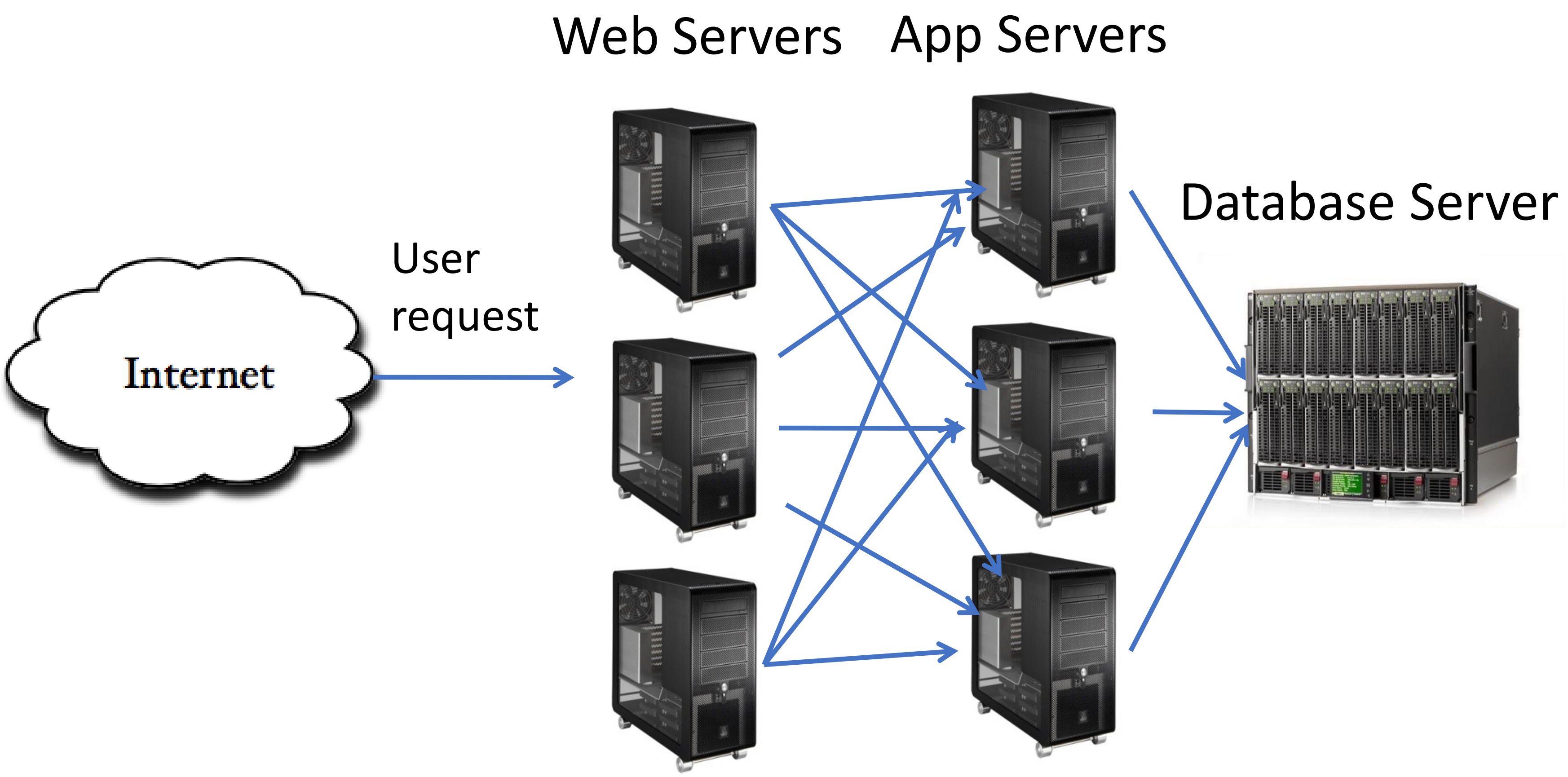


3 Tier architecture



Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*

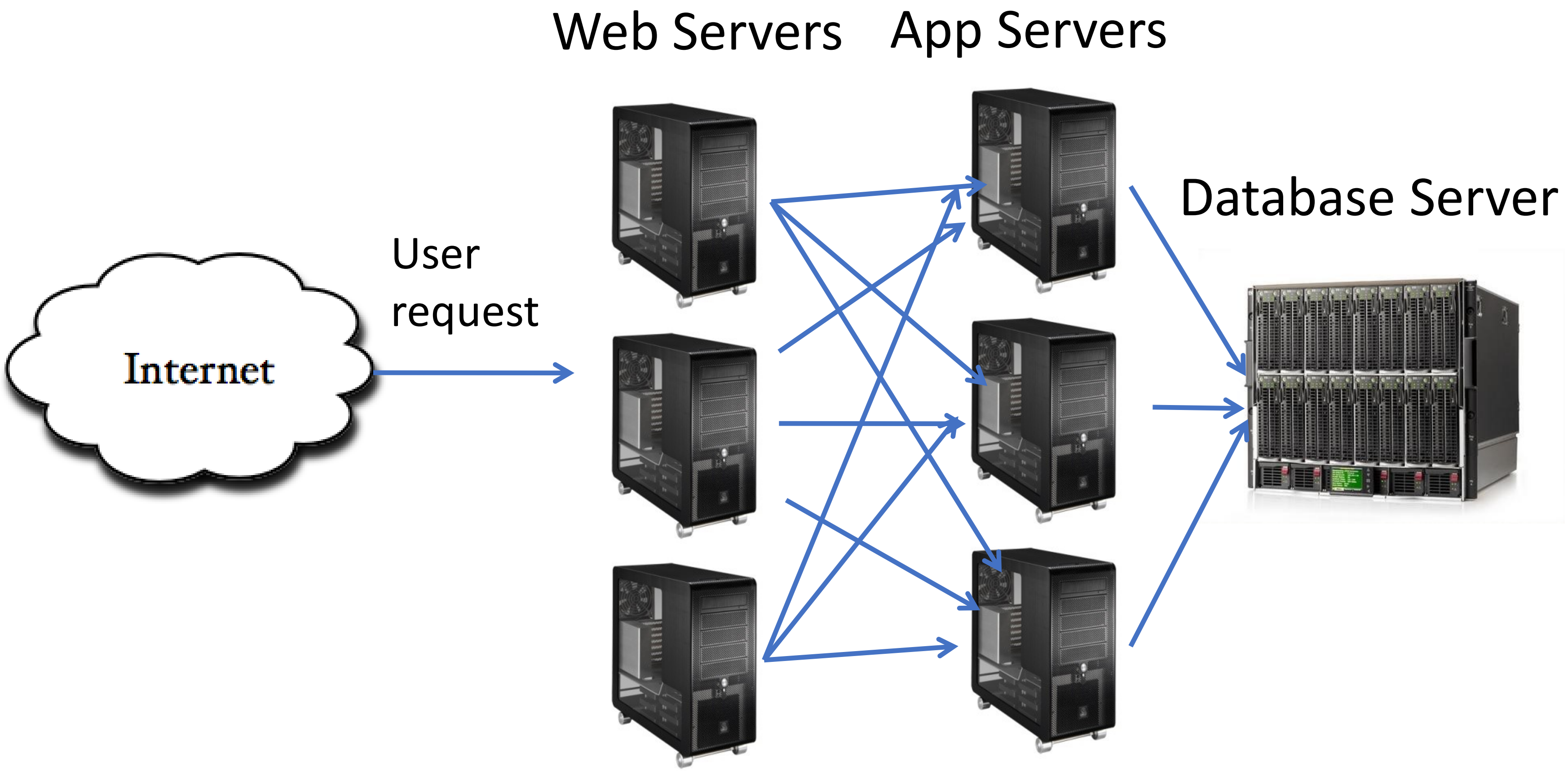
3 Tier architecture



Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*
Database Server → scales *vertically*

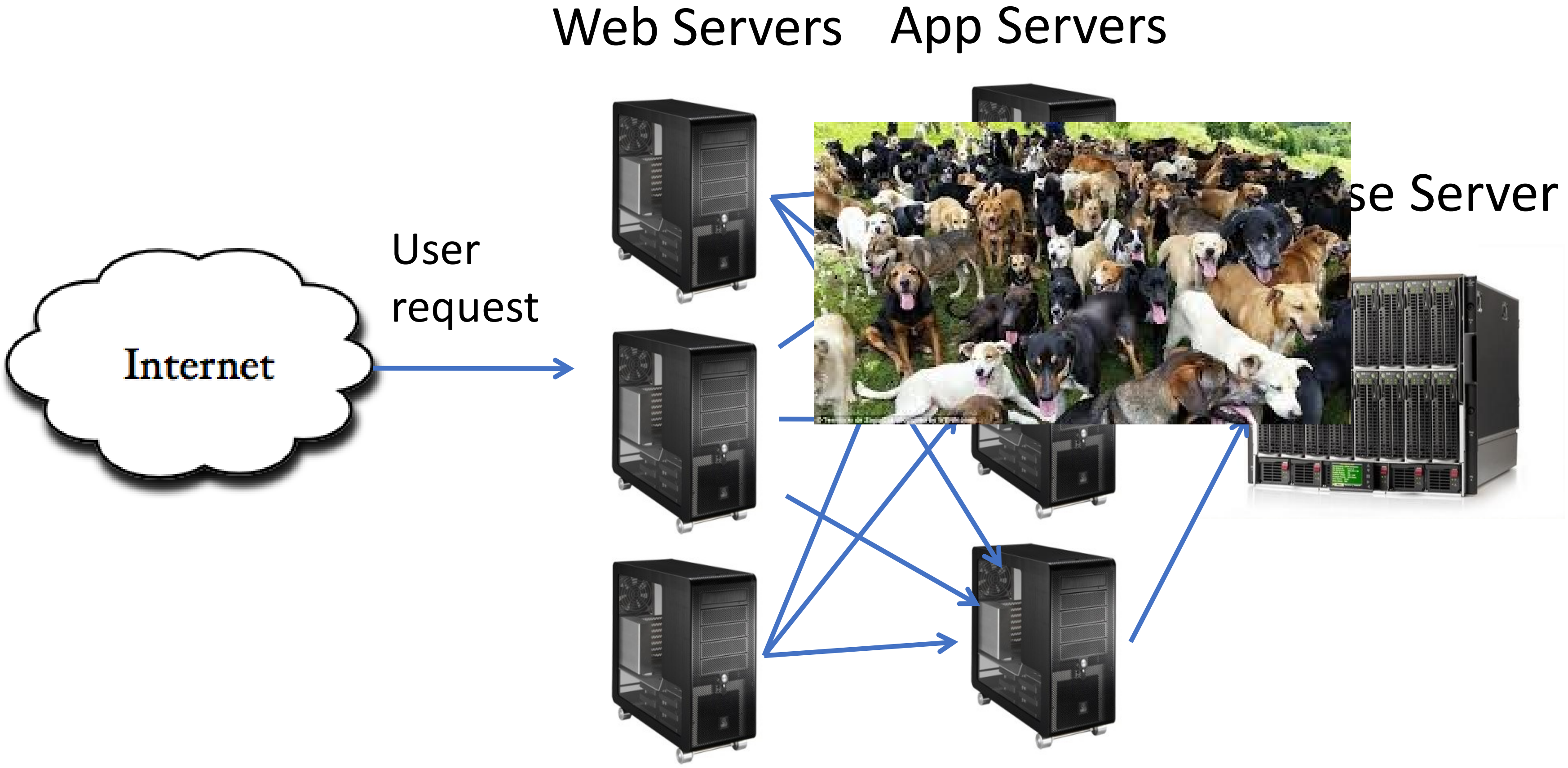
Horizontal Scale → "Shared Nothing"

3 Tier architecture



Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*
Database Server → scales *vertically*
Horizontal Scale → "Shared Nothing"
Why is this a good arrangement?

3 Tier architecture

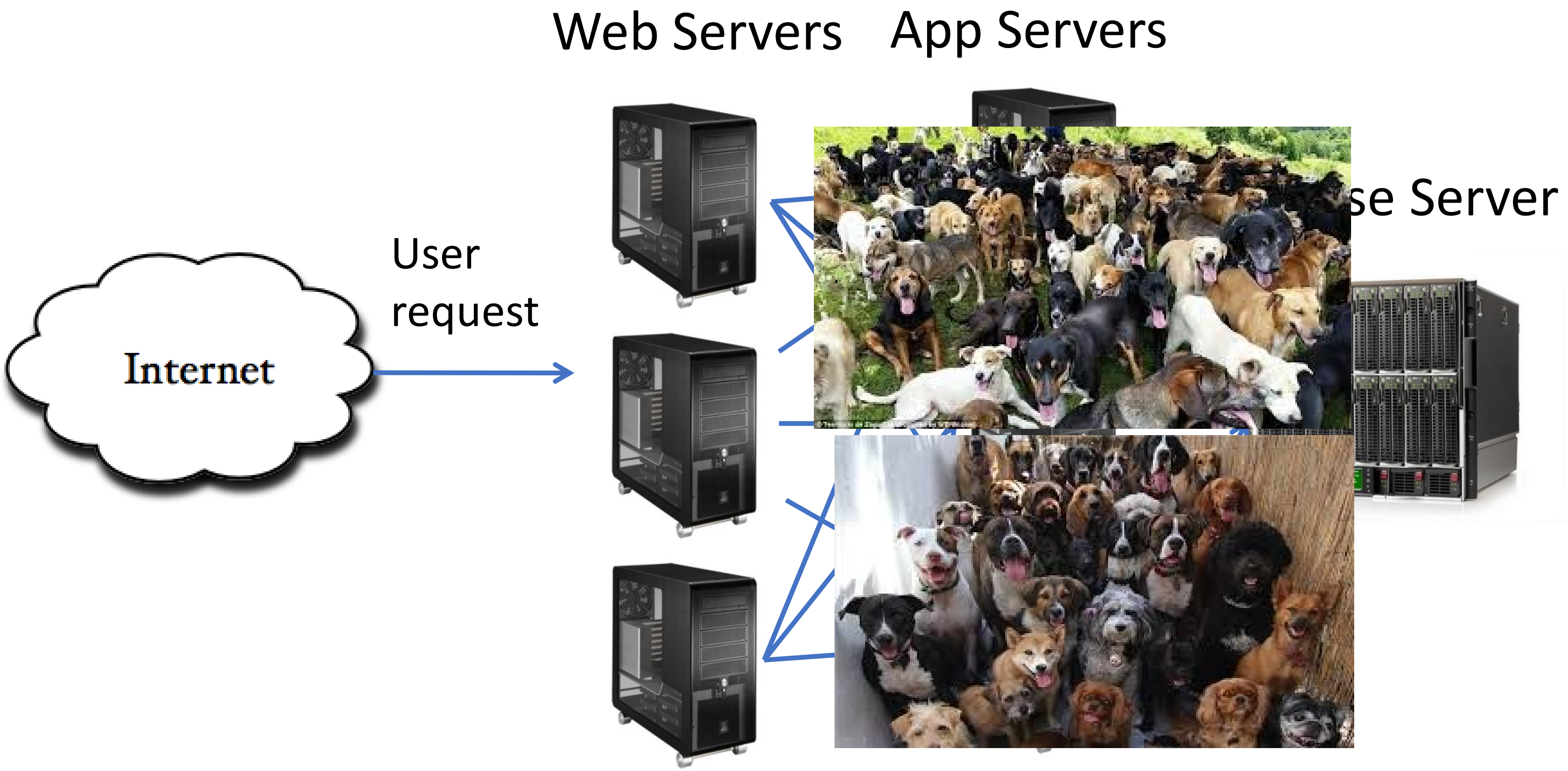


Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*
Database Server → scales *vertically*

Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

3 Tier architecture



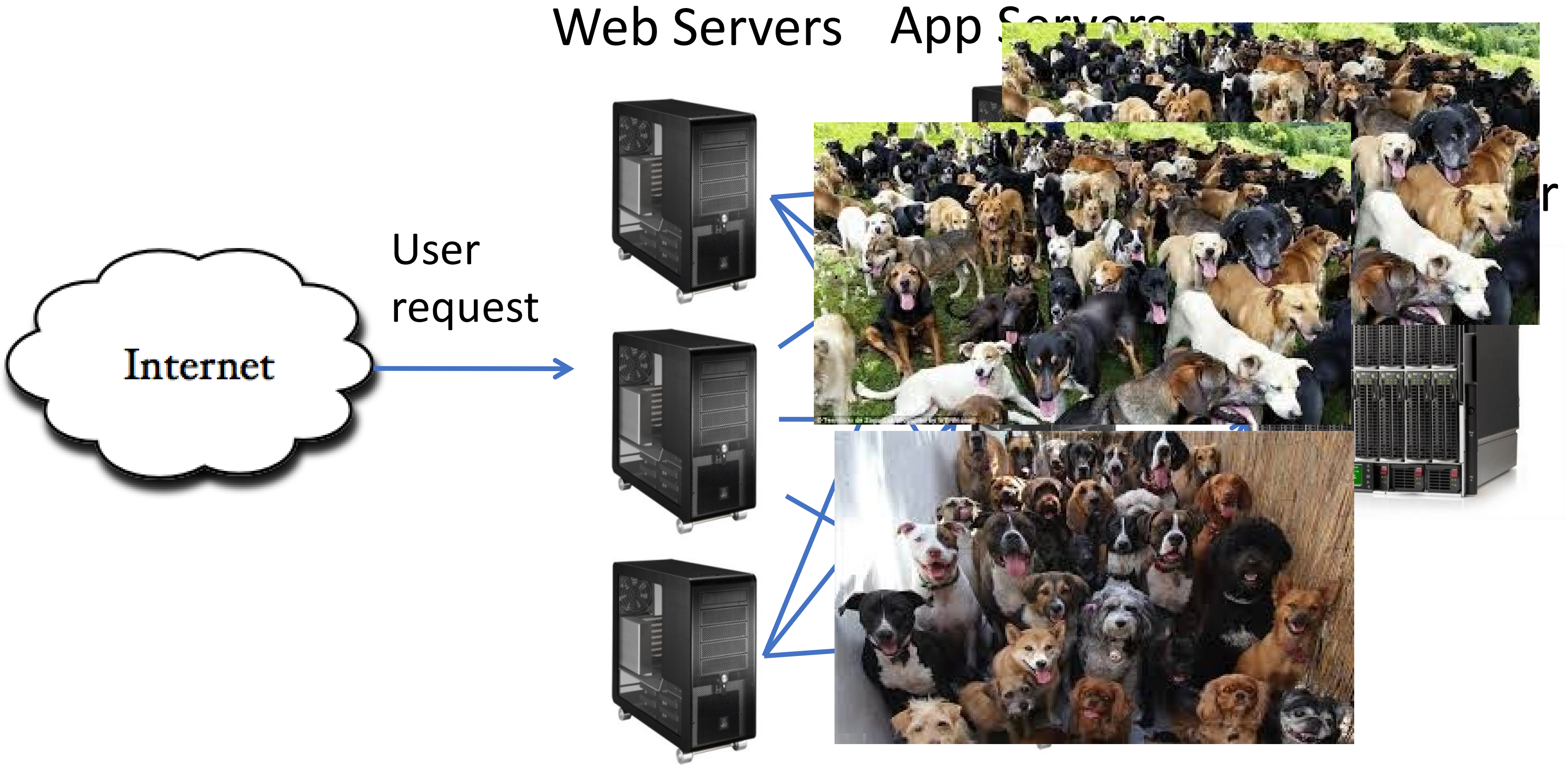
Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*

Database Server → scales *vertically*

Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

3 Tier architecture



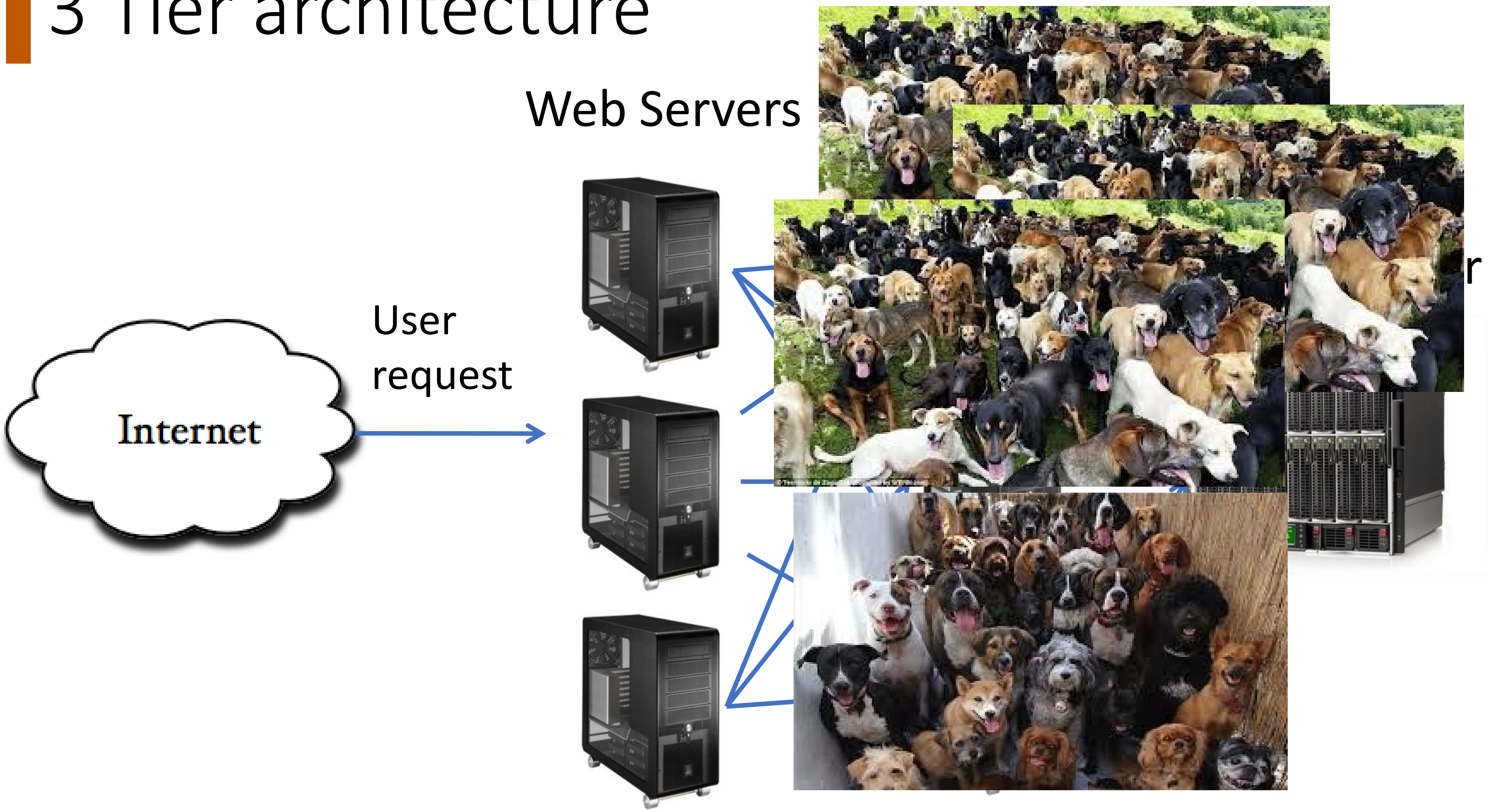
Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*

Database Server → scales *vertically*

Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

3 Tier architecture



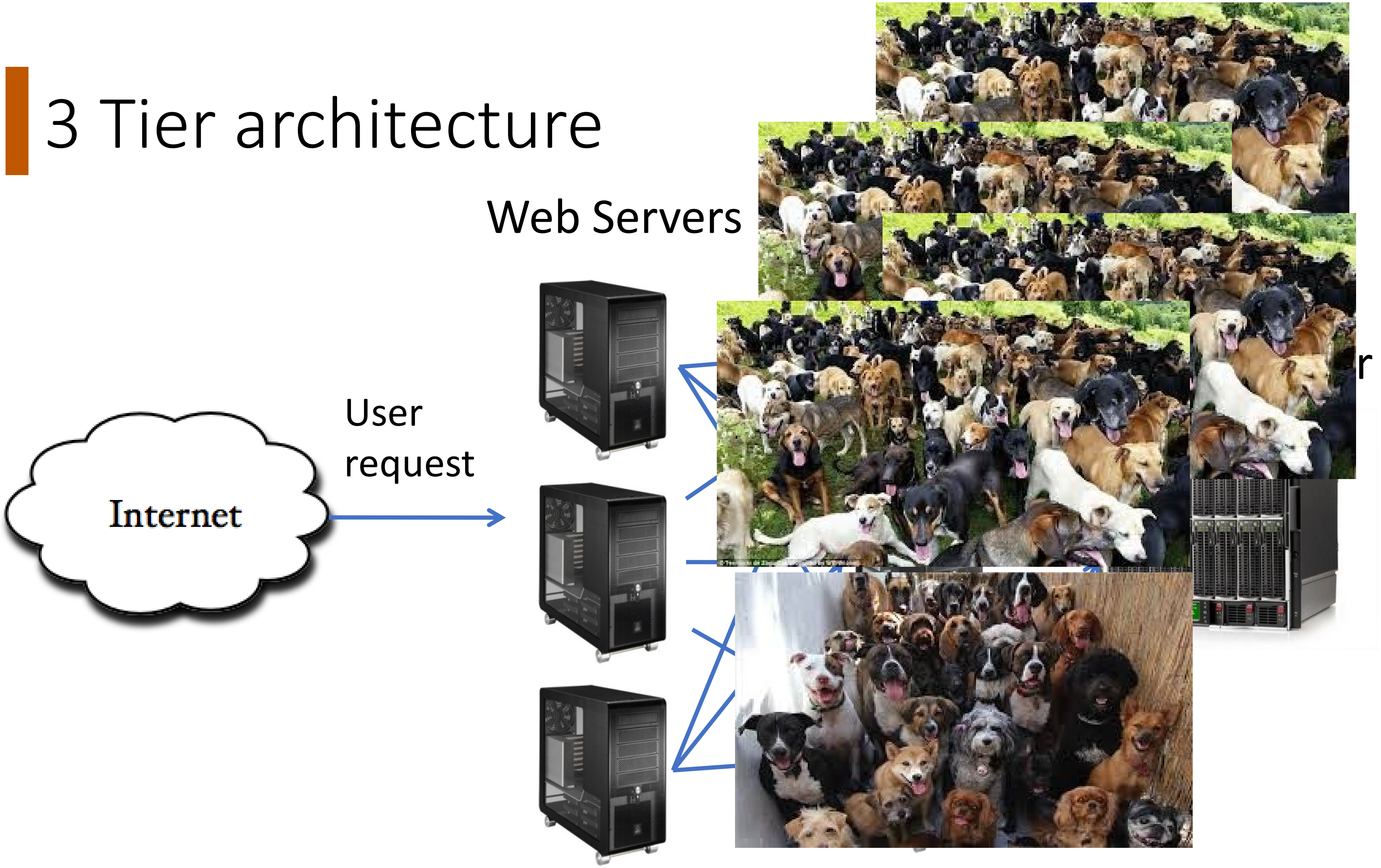
Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*

Database Server → scales *vertically*

Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

3 Tier architecture

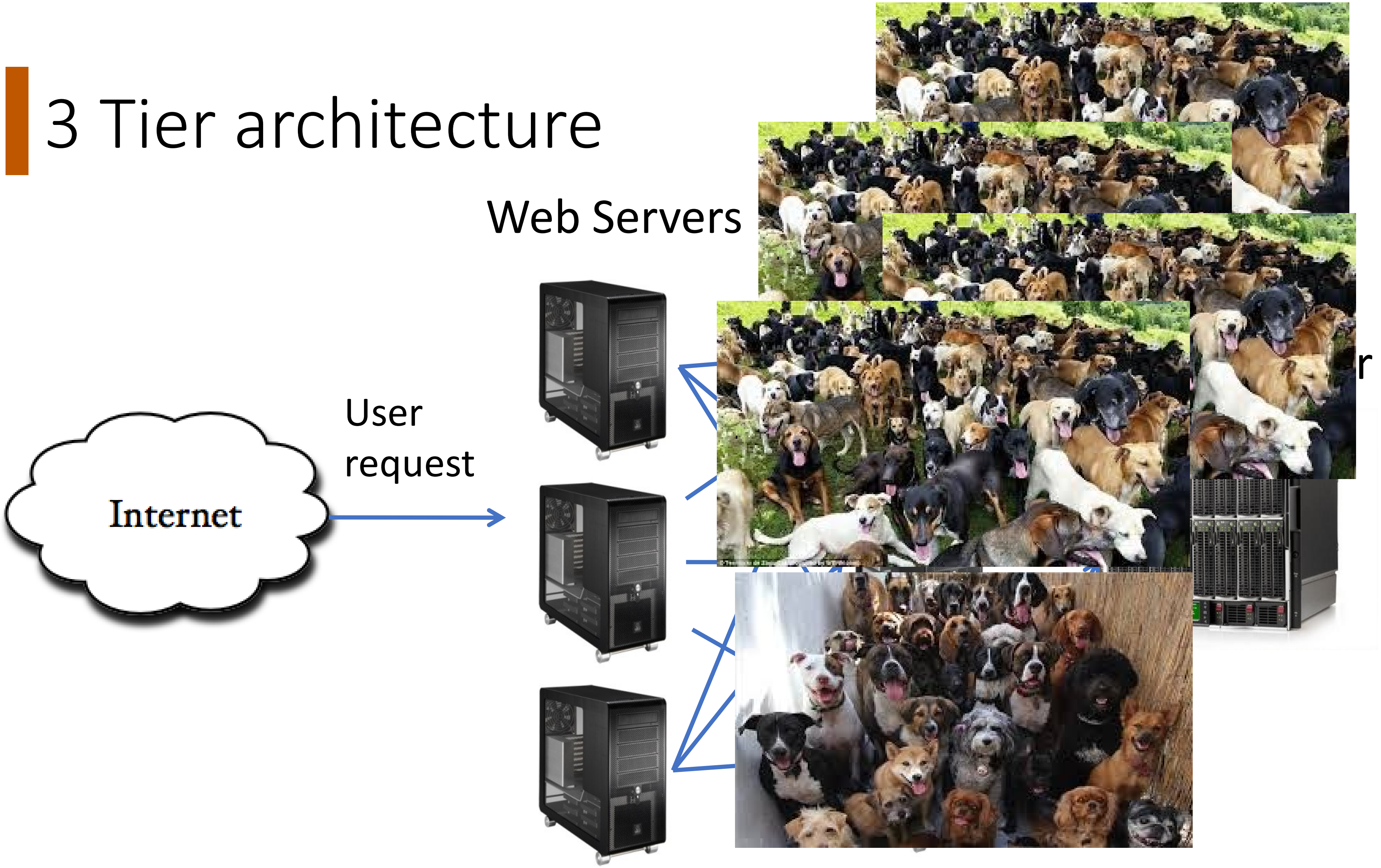


Web Servers (Presentation Tier) and App servers (Business Tier) scale *horizontally*
Database Server → scales *vertically*

Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

3 Tier architecture



Web Servers (Presentation Tier) and Ap… …izontally

Database Server → scales *vertically*

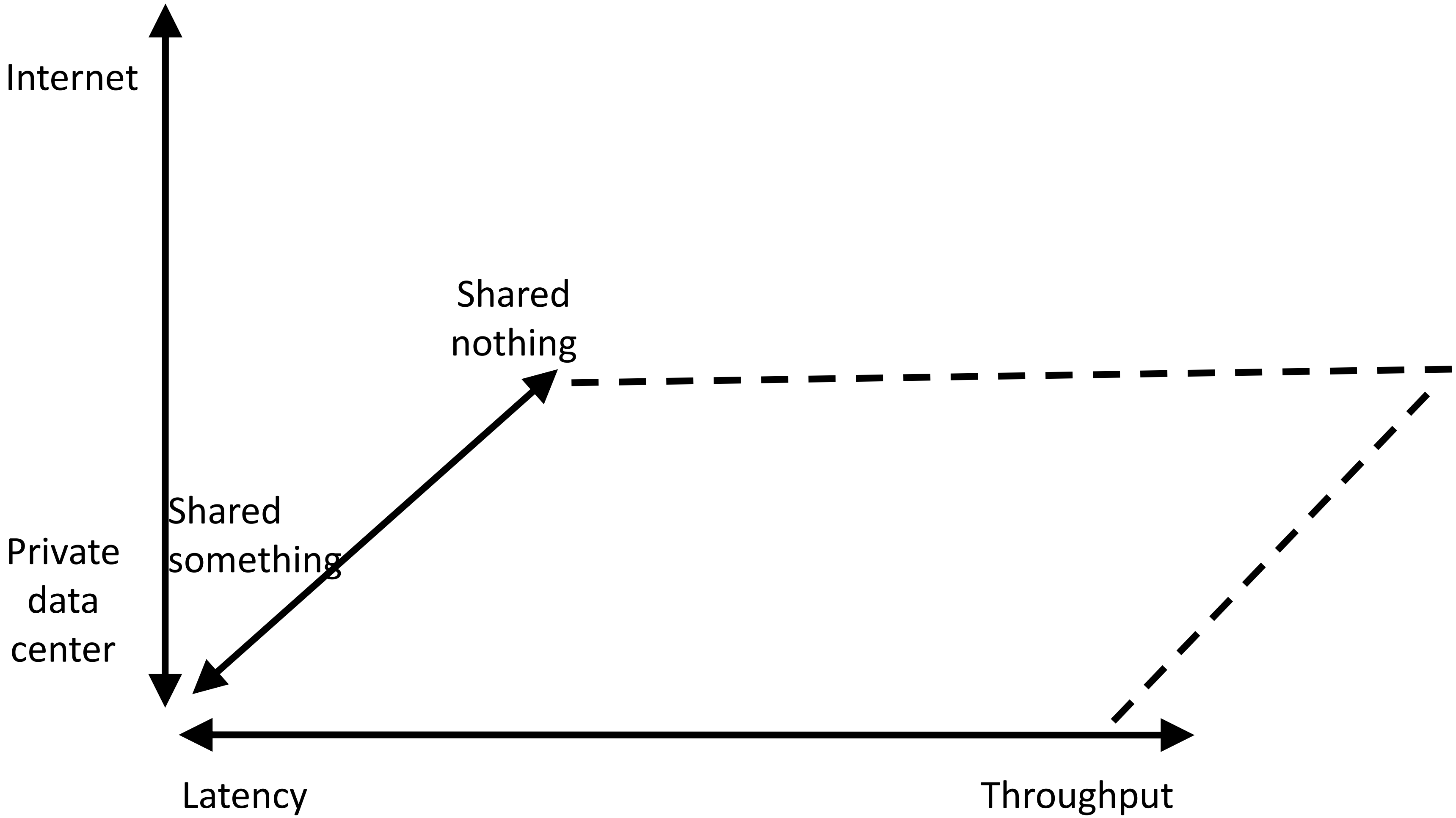
Horizontal Scale → "Shared Nothing"

Why is this a good arrangement?

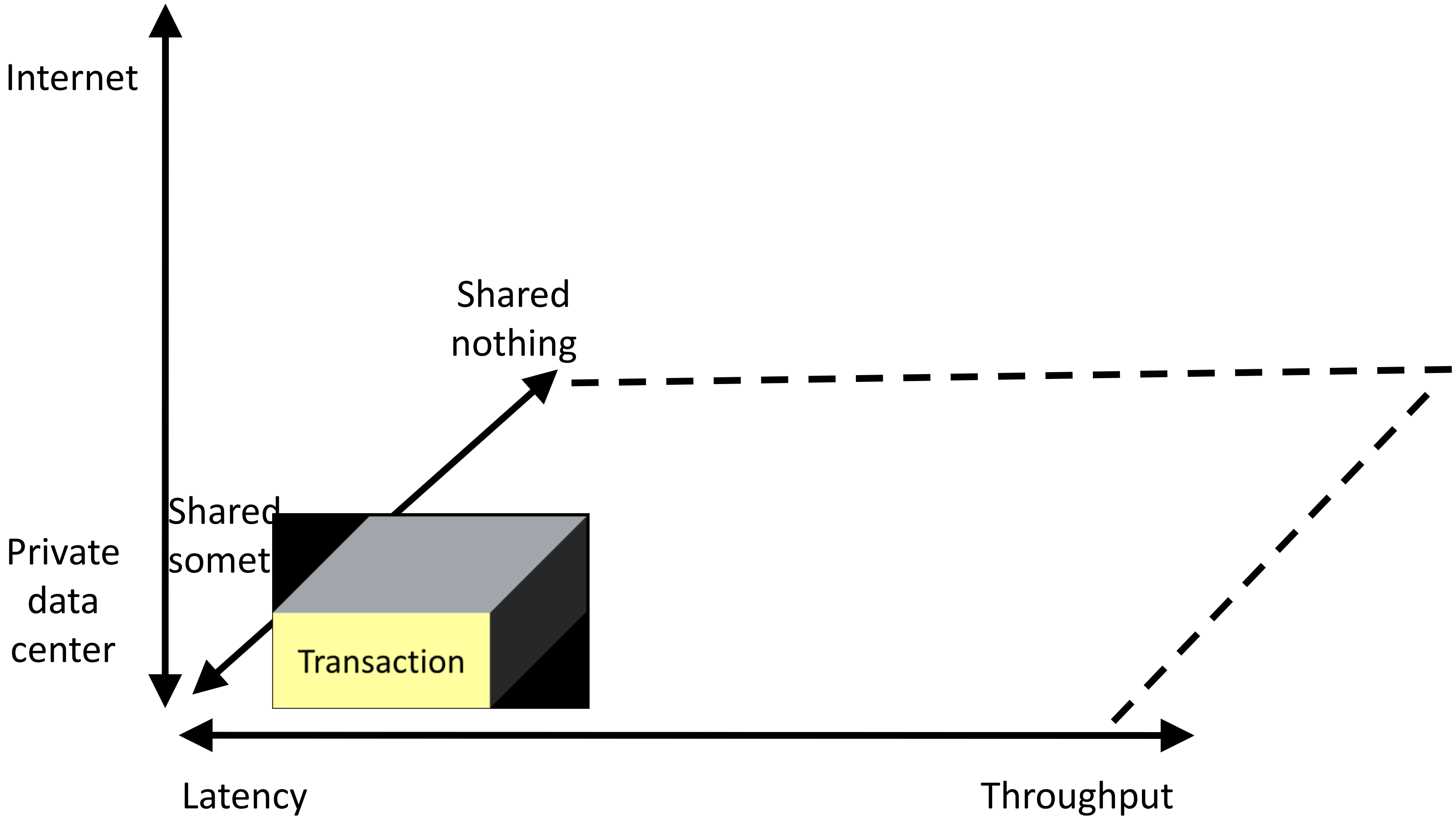
Vertical scale gets you a long way, but there is always a bigger problem size

Horizontal Scale: Goal

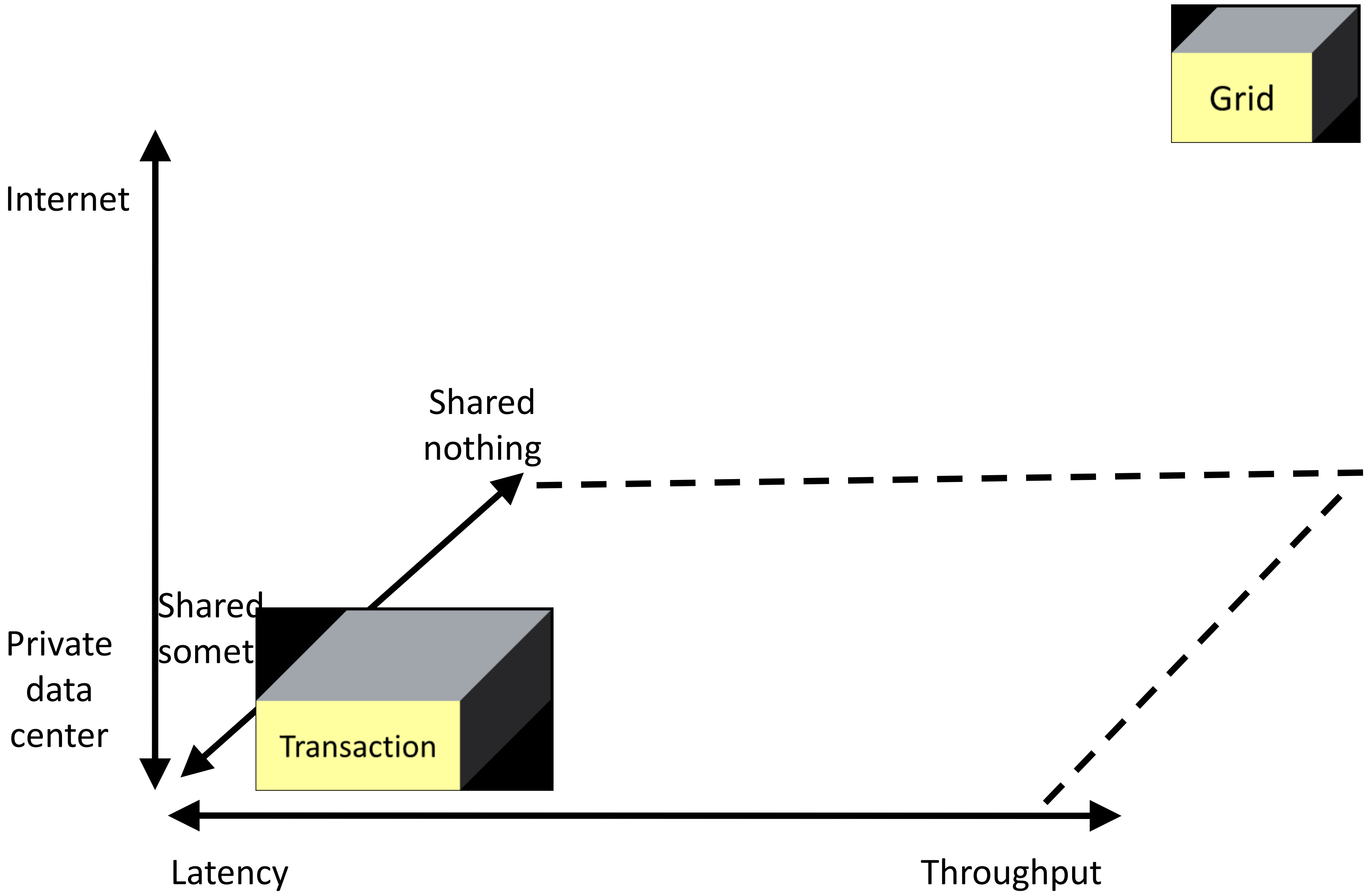
Design Space



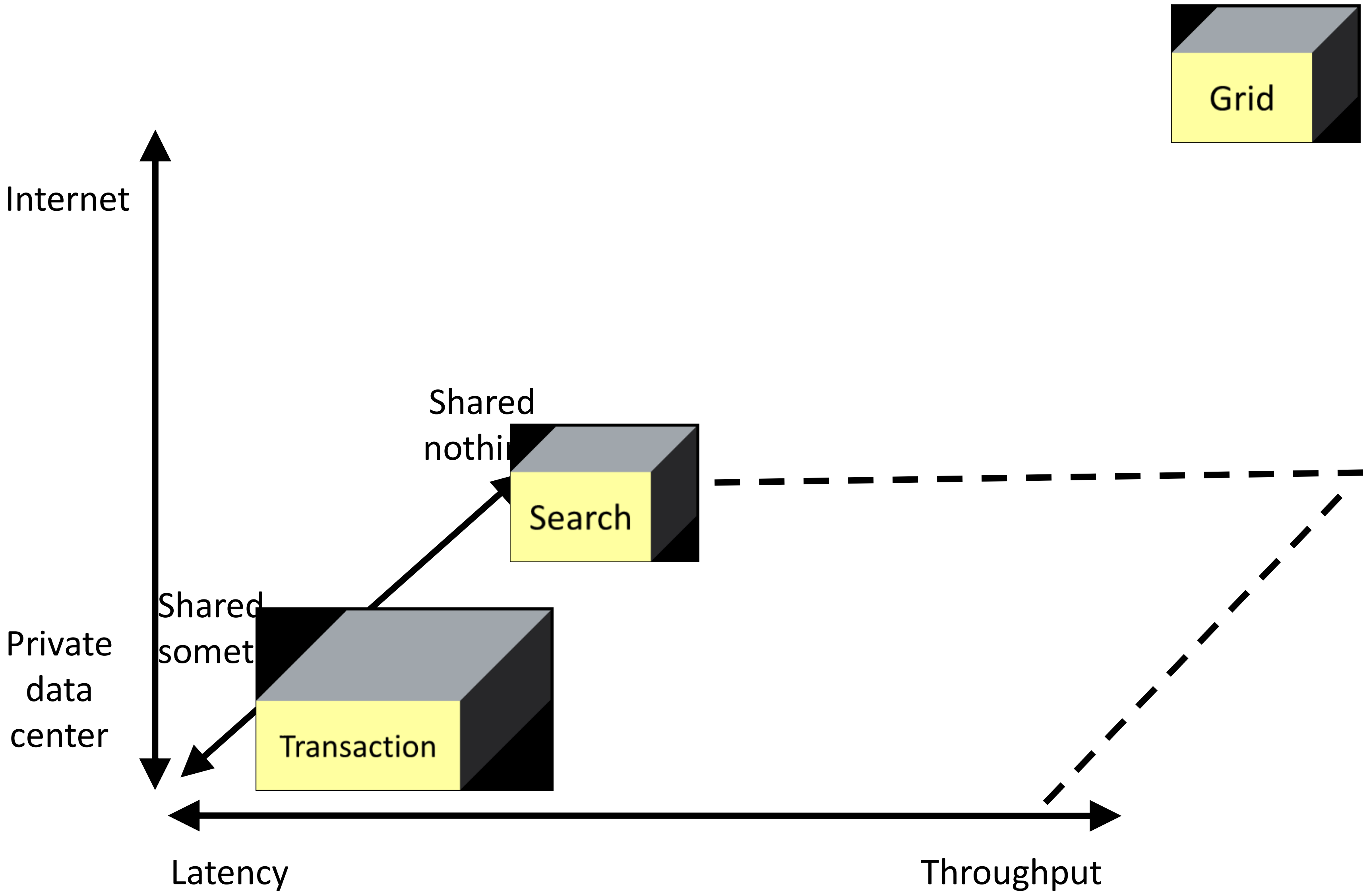
Design Space



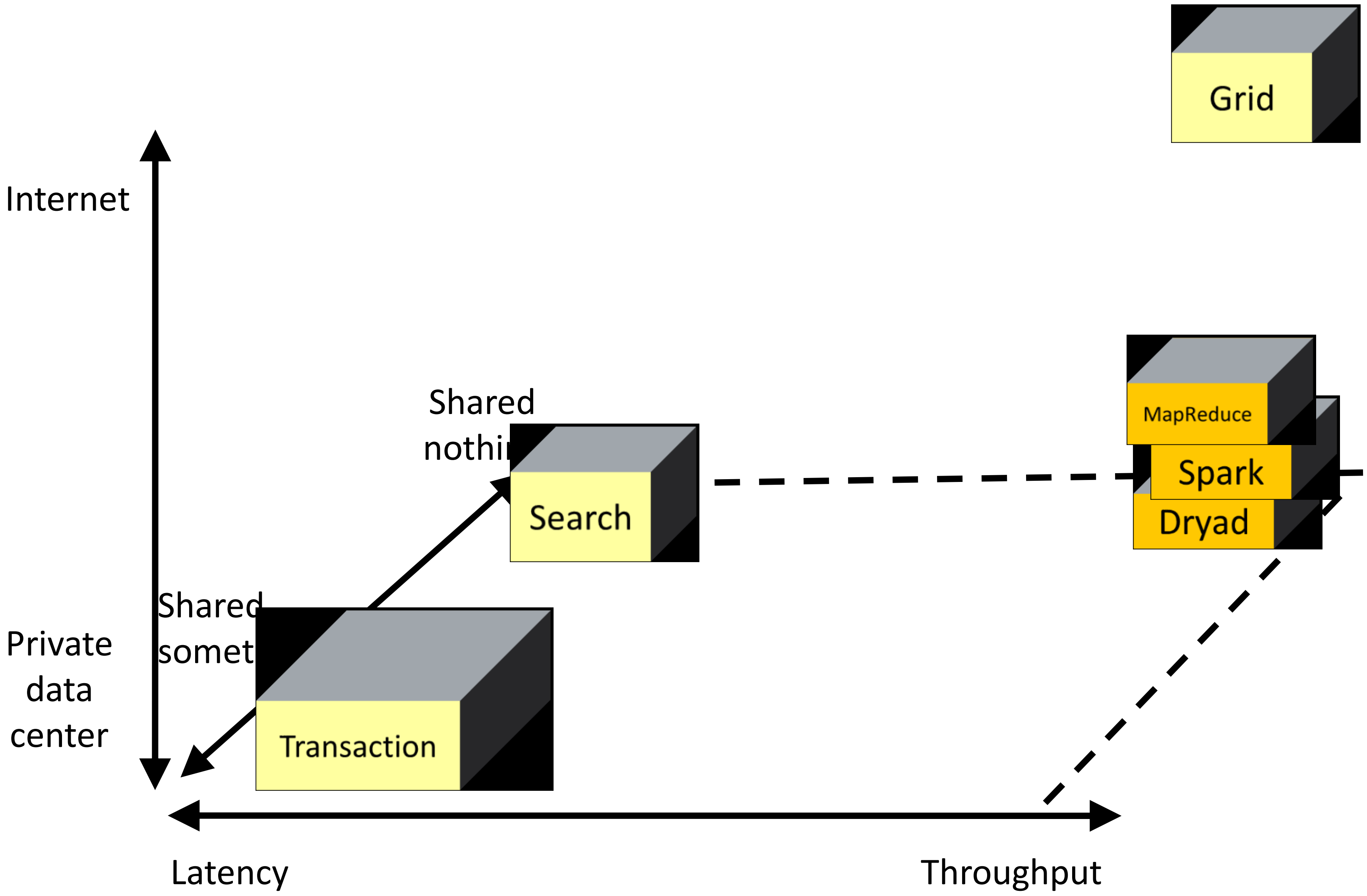
Design Space



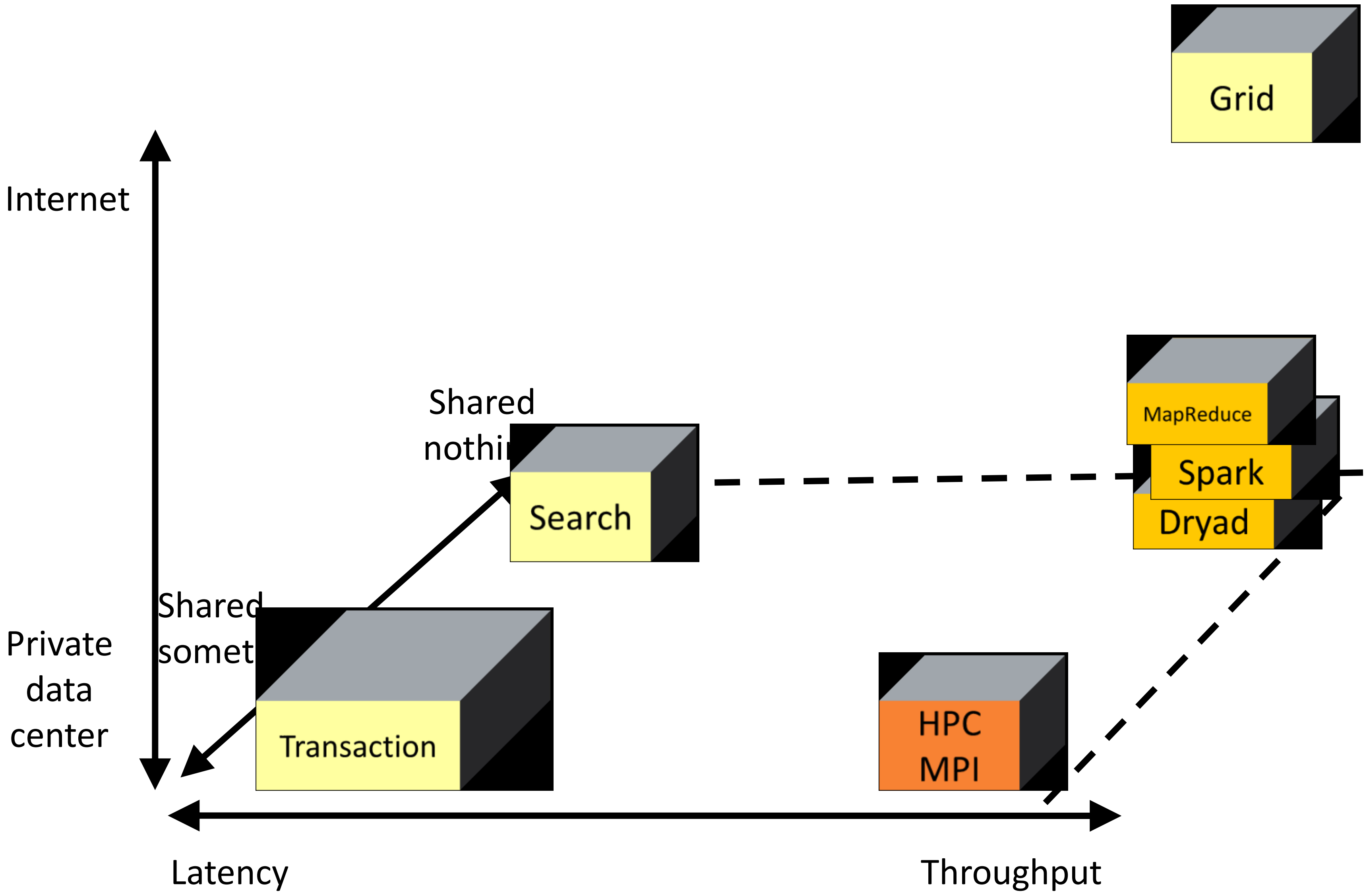
Design Space



Design Space

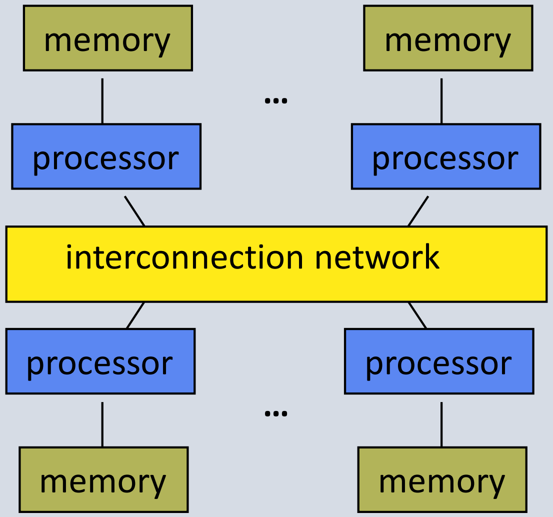


Design Space



Parallel Architectures and MPI

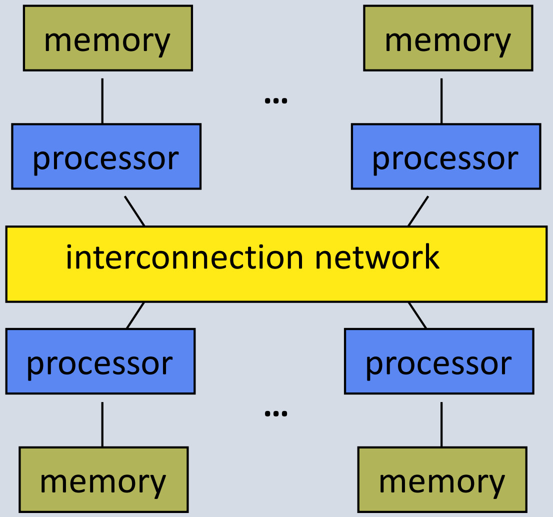
Parallel Architectures and MPI



Parallel Architectures and MPI

Distributed Memory Multiprocessor

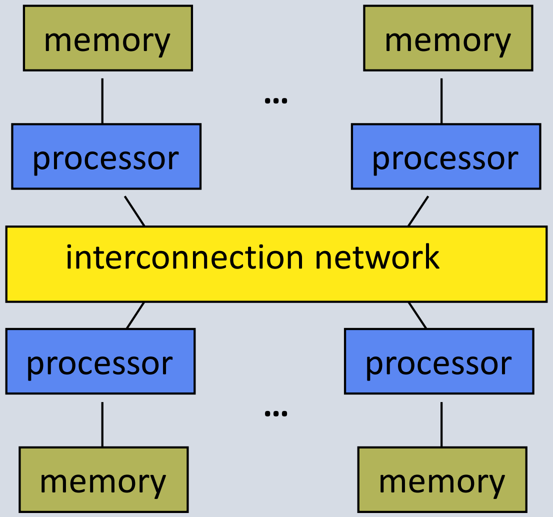
Messaging between nodes



Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes



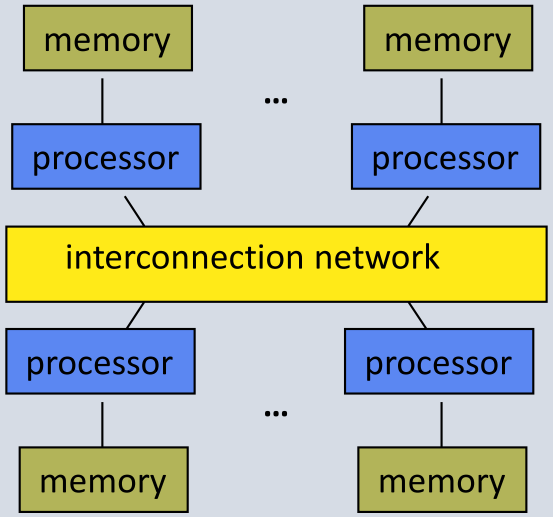
Massively Parallel Processor (MPP)

Many, many processors

Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes



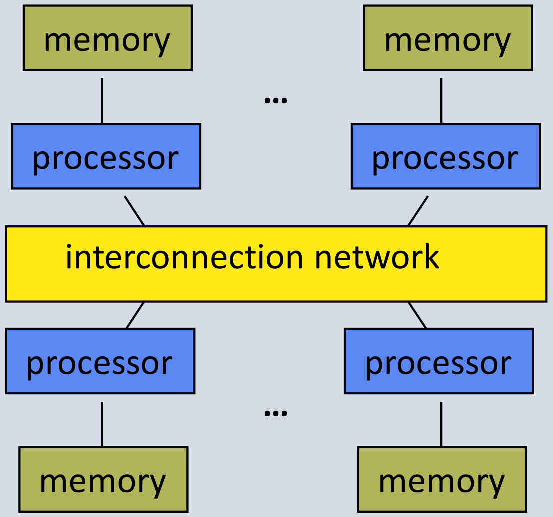
Massively Parallel Processor (MPP)

Many, many processors

Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes

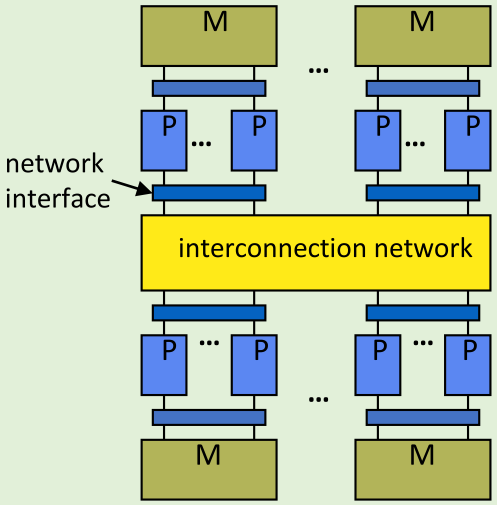


Massively Parallel Processor (MPP)

Many, many processors

Cluster of SMPs

- Shared memory in SMP node
- Messaging ←→ SMP nodes

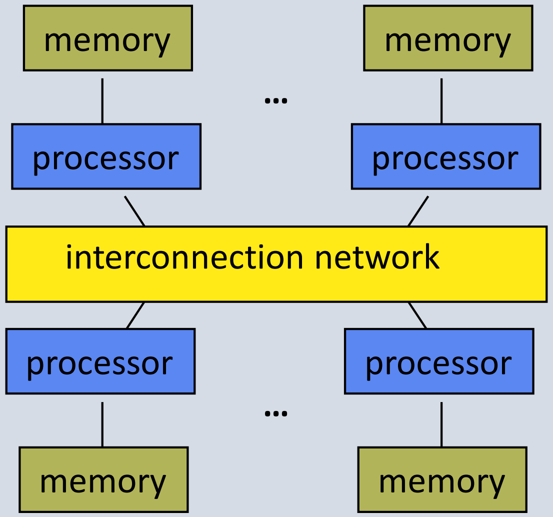


- also regarded as MPP if processor # is large

Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes

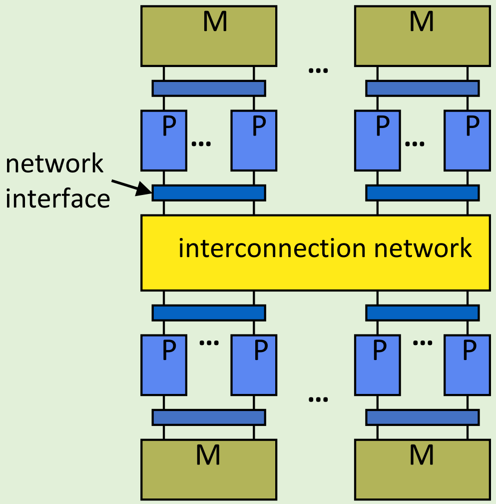


Massively Parallel Processor (MPP)

Many, many processors

Cluster of SMPs

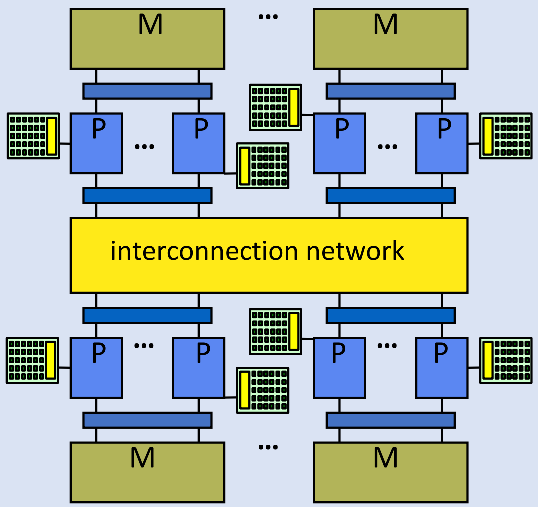
- Shared memory in SMP node
- Messaging ←→ SMP nodes



- also regarded as MPP if processor # is large

Multicore SMP+GPU Cluster

- Shared mem in SMP node
- Messaging between nodes

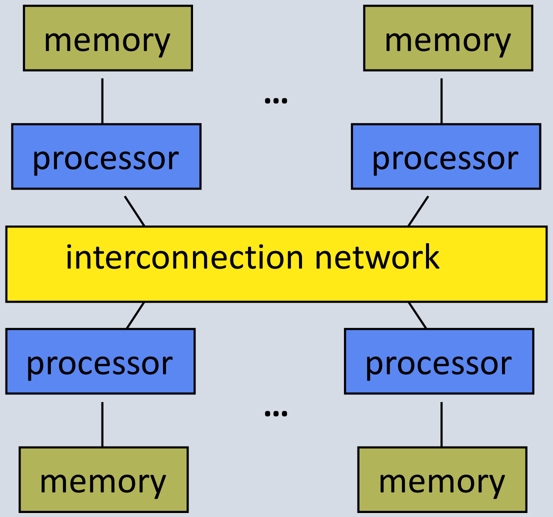


- GPU accelerators attached

Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes

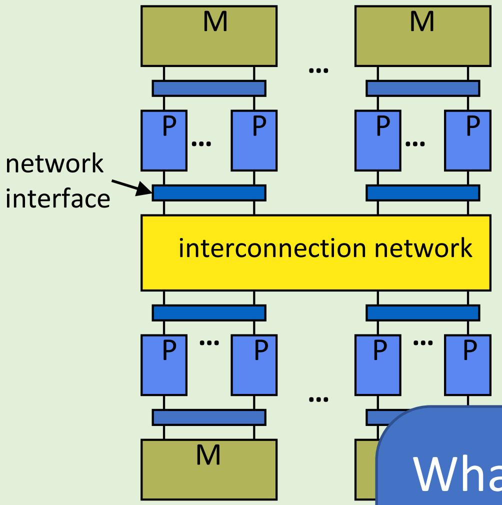


Massively Parallel Processor (MPP)

Many, many processors

Cluster of SMPs

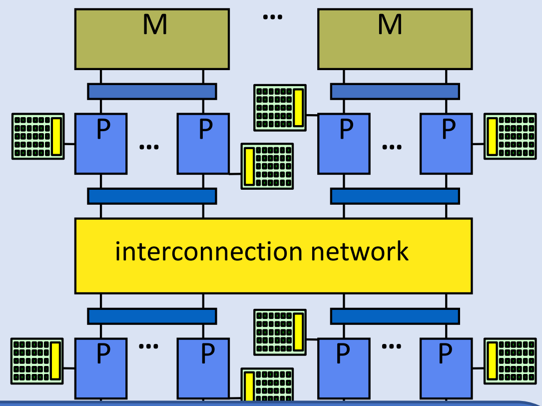
- Shared memory in SMP node
- Messaging ←→ SMP nodes



- also regarded
  processor # is

Multicore SMP+GPU Cluster

- Shared mem in SMP node
- Messaging between nodes

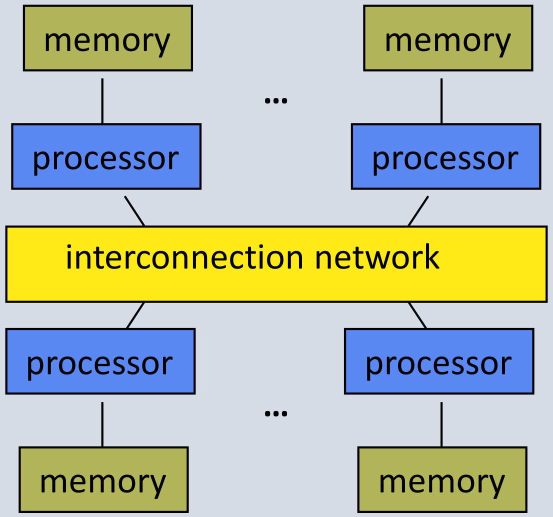


What have we left out?

Parallel Architectures and MPI

Distributed Memory Multiprocessor

Messaging between nodes

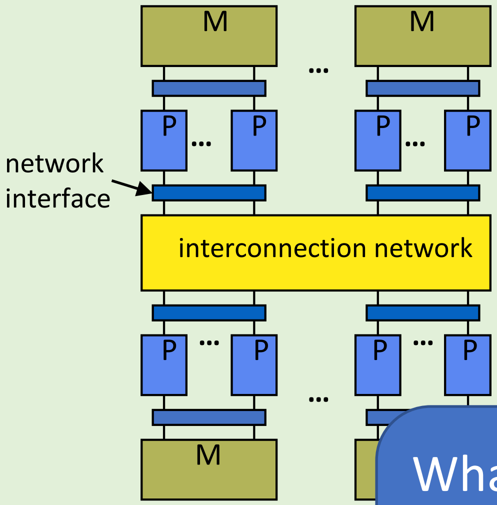


Massively Parallel Processor (MPP)

Many, many processors

Cluster of SMPs

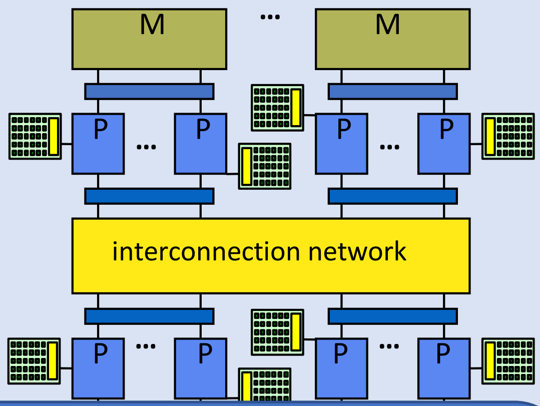
- Shared memory in SMP node
- Messaging ←→ SMP nodes



- also regarded ... processor # is ...

Multicore SMP+GPU Cluster

- Shared mem in SMP node
- Messaging between nodes



What have we left out?

- DSMs
- CMPs
- Non-GPU Accelerators

What requires extreme scale?

What requires extreme scale?

Simulations—why?

What requires extreme scale?

Simulations—why?

Simulations are sometimes more cost effective than experiments

What requires extreme scale?

Simulations—why?

Simulations are sometimes more cost effective than experiments

Why extreme scale?

More compute cycles, more memory, etc, lead for faster and/or more accurate simulations

What requires extreme scale?

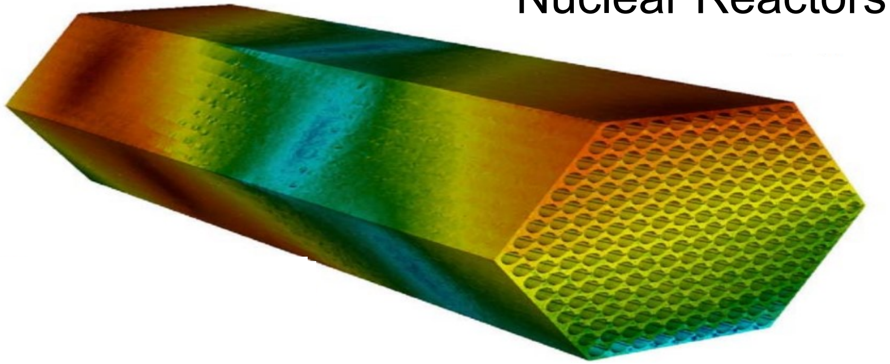
Simulations—why?

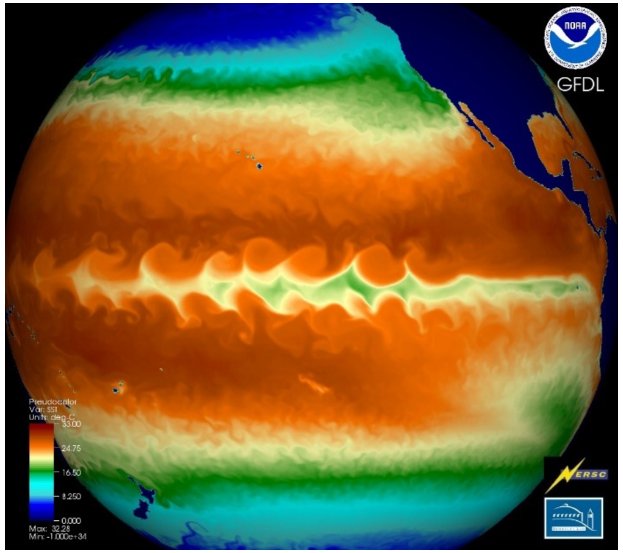
Simulations are sometimes more cost effective than experiments

Why extreme scale?

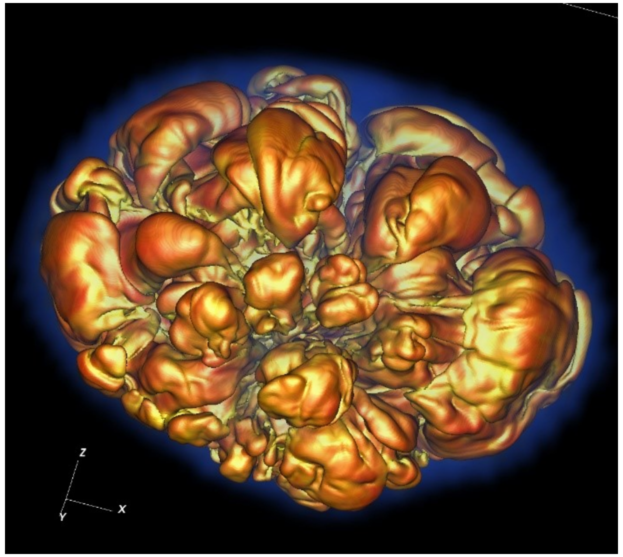
More compute cycles, more memory, etc, lead for faster and/or more accurate simulations

Nuclear Reactors





Climate Change



Astrophysics

Image credit: Prabhat, LBNL

How big is “extreme” scale?

Measured in FLOPs

FLoating point **O**perations **P**er **s**econd

- 1 GigaFLOP = 1 billion FLOPs
- 1 TeraFLOP = 1000 GigaFLOPs
- 1 PetaFLOP = 1000 TeraFLOPs
  - Most current super computers
- 1 ExaFLOP = 1000 PetaFLOPs
  - Arriving in 2018 (supposedly)

How big is "extreme" scale?

Measured in FLOPs

FLoating point **O**perations **P**er **s**econd

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
6	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL	1,572,864	17,173.2	20,132.7	7,890

How big is “extreme” scale?

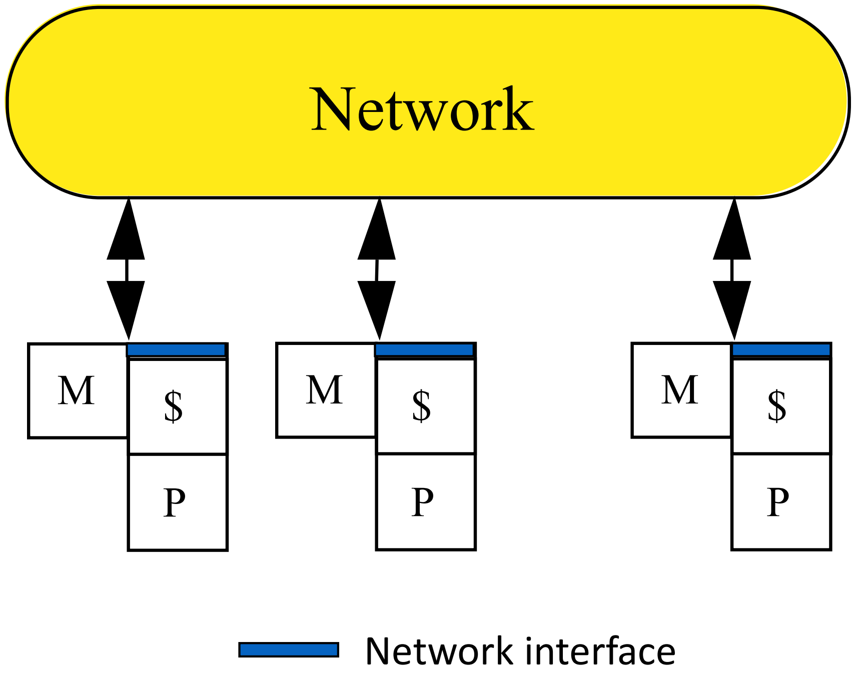
Measured in FLOPs

FLoating point **O**perations **P**er second

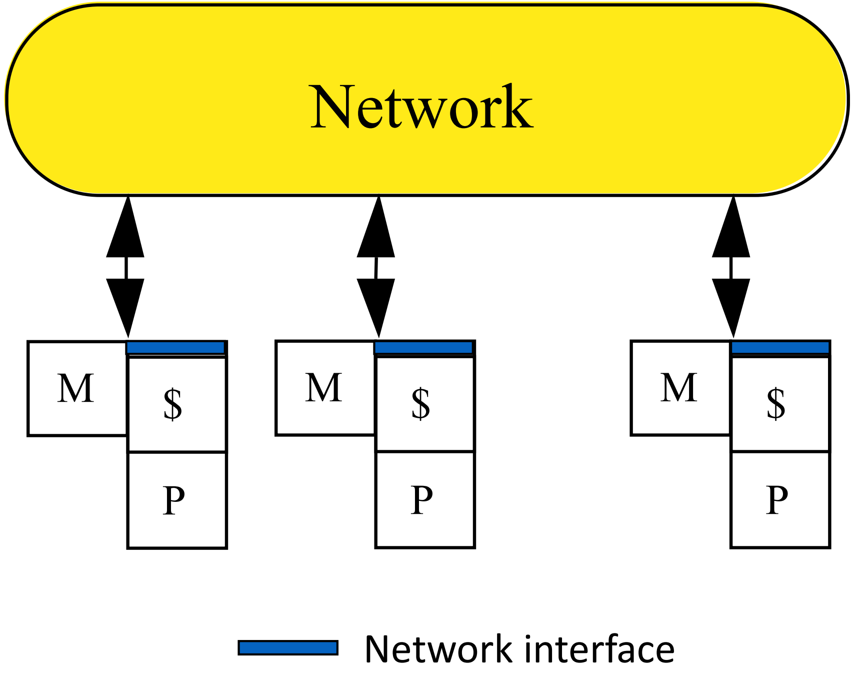
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China		014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
6	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL	1,572,864	17,173.2	20,132.7	7,890

Distributed Memory Multiprocessors

Distributed Memory Multiprocessors



Distributed Memory Multiprocessors

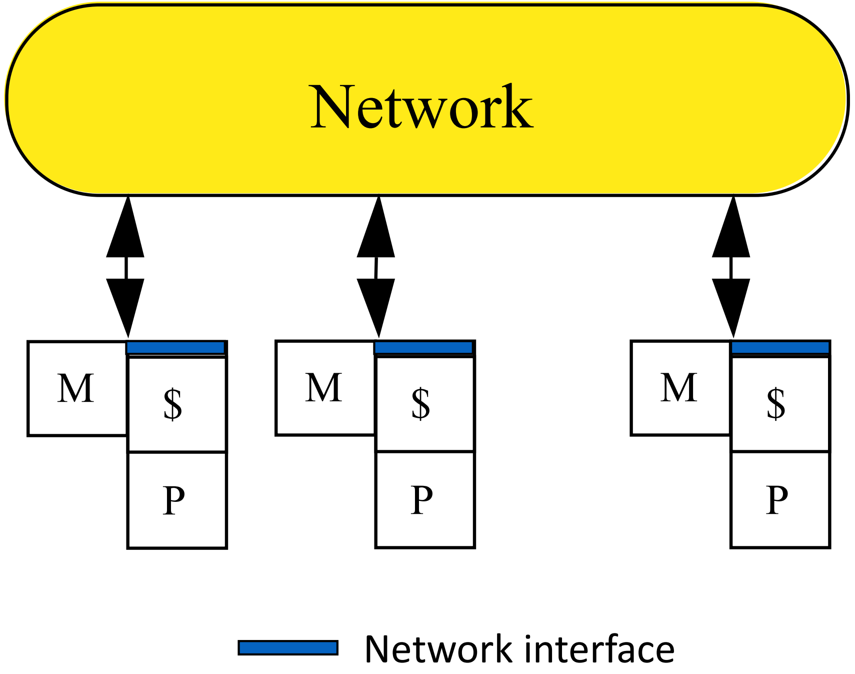


- Nodes: complete computer
  - Including I/O
- Nodes communicate via network
  - Standard networks (IP)
  - Specialized networks (RDMA, fiber)

Distributed Memory Multiprocessors

Each processor has a local memory

Physically separated address space



- Nodes: complete computer
  - Including I/O
- Nodes communicate via network
  - Standard networks (IP)
  - Specialized networks (RDMA, fiber)

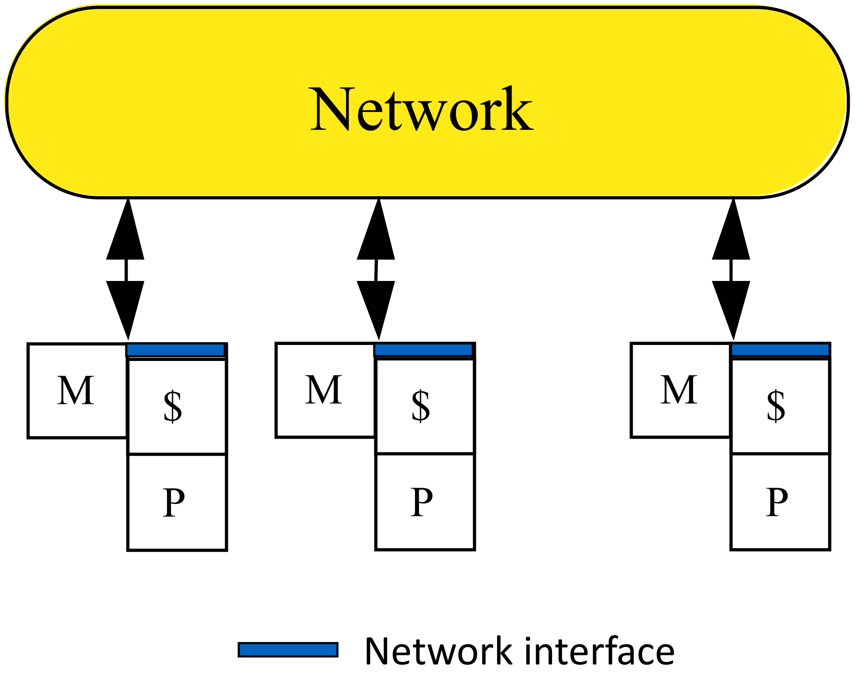
Distributed Memory Multiprocessors

Each processor has a local memory

- Physically separated address space

Processors communicate to access non-local data

- Message communication
  - *Message passing architecture*
- Processor interconnection network



- Nodes: complete computer
  - Including I/O
- Nodes communicate via network
  - Standard networks (IP)
  - Specialized networks (RDMA, fiber)

Distributed Memory Multiprocessors

Each processor has a local memory

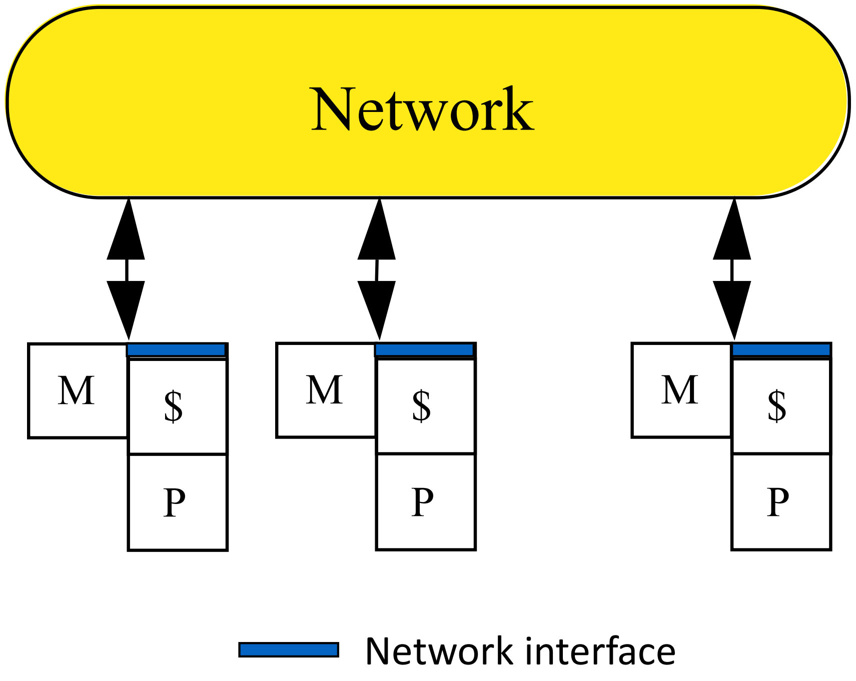
- Physically separated address space

Processors communicate to access non-local data

- Message communication
  - *Message passing architecture*
- Processor interconnection network

Parallel applications partitioned across

- Processors: execution units
- Memory: data partitioning



- Nodes: complete computer
  - Including I/O
- Nodes communicate via network
  - Standard networks (IP)
  - Specialized networks (RDMA, fiber)

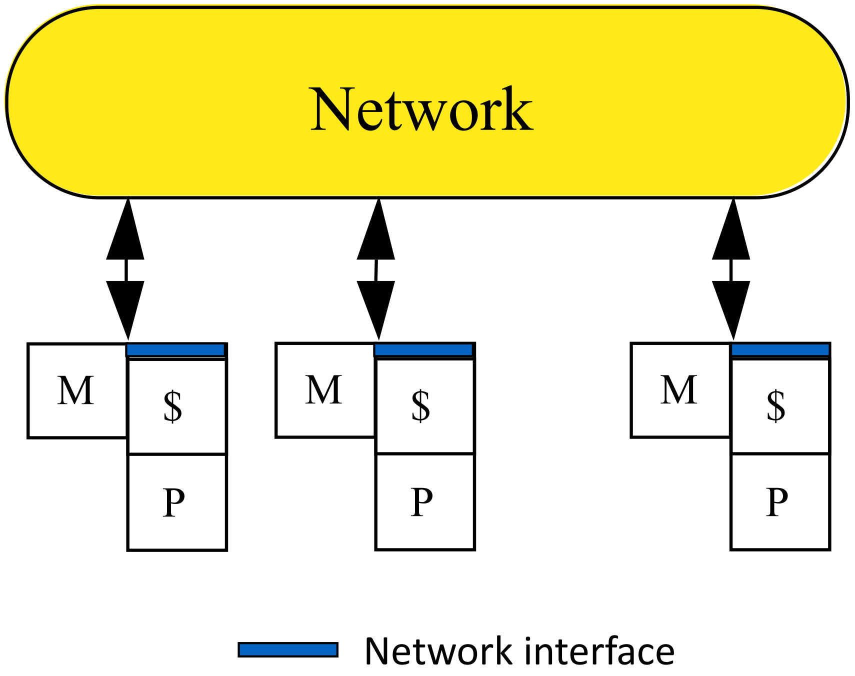
Distributed Memory Multiprocessors

Each processor has a local memory
- Physically separated address space

Processors communicate to access non-local data
- Message communication
  - *Message passing architecture*
- Processor interconnection network

Parallel applications partitioned across
- Processors: execution units
- Memory: data partitioning

Scalable architecture
- Incremental cost to add hardware (cost of node)



- Nodes: complete computer
  - Including I/O
- Nodes communicate via network
  - Standard networks (IP)
  - Specialized networks (RDMA, fiber)

Performance: Latency and Bandwidth

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and processor
- Network interface speed vs. network bisection bandwidth

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and proce
- Network interface speed vs. network bisecti
bandwidth

Wait…bisection bandwidth?

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and proce
- Network interface speed vs. network bisecti bandwidth

Wait…bisection bandwidth?

if network is **bisected**, **bisection bandwidth == bandwidth**

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and proce
- Network interface speed vs. network bisecti
  bandwidth

Wait…bisection bandwidth?

if network is **bisected**, **bisection bandwidth** == **bandwidth** between the two partitions

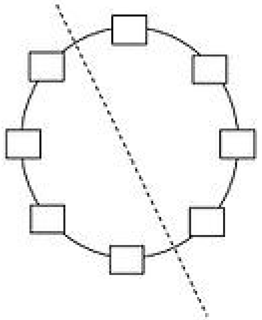
Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and proce
- Network interface speed vs. network bisecti bandwidth

Wait…bisection bandwidth?

if network is **bisected**, **bisection bandwidth** == **bandwidth** between the two partitions



Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and processor
- Network interface speed vs. network bisection bandwidth

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and processor
- Network interface speed vs. network bisection bandwidth

Latency

- Performance affected: processor may have to wait
- Hard to overlap communication and computation
- Overhead to communicate: a problem in many machines

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and processor
- Network interface speed vs. network bisection bandwidth

Latency

- Performance affected: processor may have to wait
- Hard to overlap communication and computation
- Overhead to communicate: a problem in many machines

Latency hiding

- Increases programming system burden
- E.g.: communication/computation overlap, prefetch

Performance: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Match limits in network, memory, and processor
- Network interface speed vs. network bisection bandwidth

Latency

- Performance affected: processor may have to wait
- Hard to overlap communication and computation
- Overhead to communicate: a problem in many machines

Latency hiding

- Increases programming system burden
- E.g.: communication/computation overlap, prefetch

Is this different from metrics we've cared about so far?

Ostensible Advantages of Distributed Memory Architectures

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Communication explicit, simpler to understand

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Communication explicit, simpler to understand

Explicit communication →

focus attention on costly aspect of parallel computation

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Communication explicit, simpler to understand

Explicit communication →

- focus attention on costly aspect of parallel computation

Synchronization →

- naturally associated with sending messages
- reduces possibility for errors from incorrect synchronization

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Communication explicit, simpler to understand

Explicit communication →

- focus attention on costly aspect of parallel computation

Synchronization →

- naturally associated with sending messages
- reduces possibility for errors from incorrect synchronization

Easier to use sender-initiated communication →

- some advantages in performance

Ostensible Advantages of Distributed Memory Architectures

Hardware simpler (especially versus NUMA), more scalable

Communication explicit, simpler to understand

Explicit communication →

focus attention on costly aspect of parallel computation

Synchronization →

naturally associated with sending messages

reduces possibility for errors from incorrect synchronization

Easier to use sender-initiated communication →

some advantages in performance

Can you think of any *disadvantages*?

Running on Supercomputers

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Supercomputers used continuously

Processors: “scarce resource”

jobs are “plentiful”

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Supercomputers used continuously

Processors: “scarce resource”

jobs are “plentiful”

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Supercomputers used continuously

Processors: “scarce resource”

jobs are “plentiful”

- Scheduler runs scripts that initialize the environment
  - Typically done with environment variables

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Supercomputers used continuously

Processors: “scarce resource”

jobs are “plentiful”

- Scheduler runs scripts that initialize the environment
  - Typically done with environment variables
- At the end of initialization, it is possible to infer:
  - What the desired job configuration is (i.e., how many tasks per node)
  - What other nodes are involved
  - How your node’s tasks relates to the overall program

Running on Supercomputers

- Programmer plans a ***job***; job ==
  - parallel binary program
  - “input deck” (specifies input data)
- Submit job to a ***queue***
- Scheduler allocates resources when
  - resources are available,
  - (or) the job is deemed “high priority”

Sometimes 1 job takes whole machine

These are called “hero runs”…

Sometimes many smaller jobs

Supercomputers used continuously

Processors: “scarce resource”

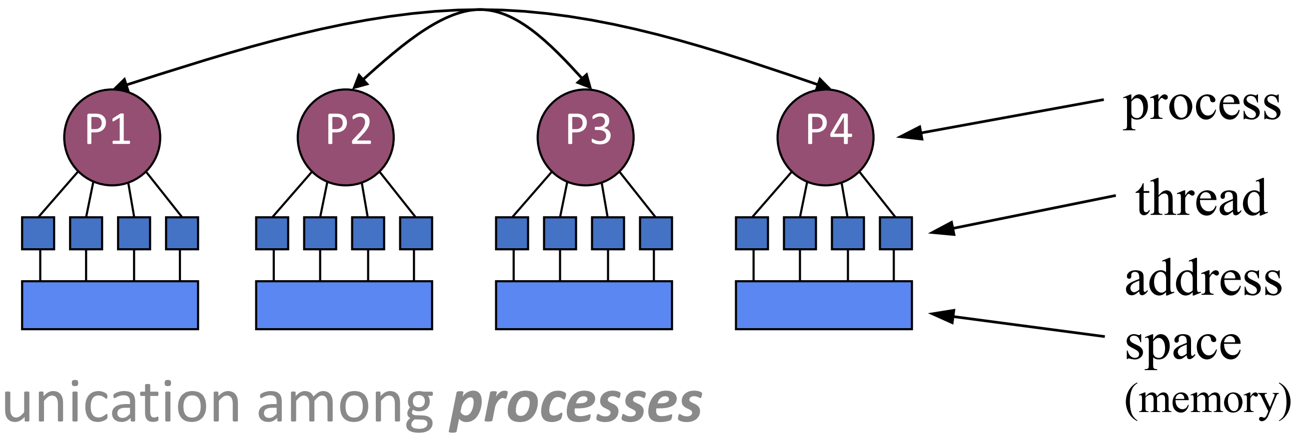
jobs are “plentiful”

- Scheduler runs scripts that initialize the environment
  - Typically done with environment variables
- At the end of initialization, it is possible to infer:
  - What the desired job configuration is (i.e., how many tasks per node)
  - What other nodes are involved
  - How your node’s tasks relates to the overall program
- MPI library interprets this information, hides the details

The Message-Passing Model

Process: a program counter and address space

Processes: multiple threads sharing a single address space



MPI is for communication among ***processes***

- Not threads

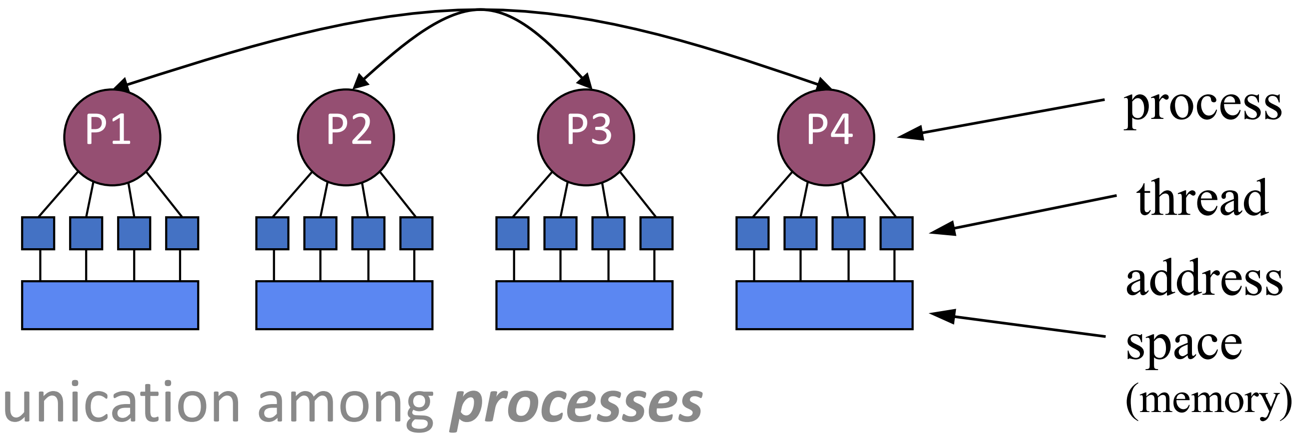
Inter-process communication consists of

- Synchronization
- Data movement

The Message-Passing Model

Process: a program counter and address space

Processes: multiple threads sharing a single address space



MPI is for communication among ***processes***

- Not threads

Inter-process communication consists of

- Synchronization
- Data movement

How does this compare with CSP?

The Message-Passing Model

Process: a program counter and address space

Proc

MPI

Inte

CSP?

The Message-Passing Model

Process: a program counter and address space

Proc

MPI

Inte

- MPI == ***M**essage-**P**assing **I**nterface specification*
  - Extended message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or product

CSP?

The Message-Passing Model

Process: a program counter and address space

Proc

- MPI == ***Message-Passing Interface specification***
  - Extended message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or product
- Specified in C, C++, Fortran 77, F90

MPI

Inte

CSP?

The Message-Passing Model

Process: a program counter and address space

Proc

- MPI == ***M**essage-**P**assing **I**nterface specification*
  - Extended message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or product
- Specified in C, C++, Fortran 77, F90
- Message Passing Interface (MPI) Forum
  - http://www.mpi-forum.org/
  - http://www.mpi-forum.org/docs/docs.html

MPI

Inte

CSP?

The Message-Passing Model

Process: a program counter and address space

Proc

MPI

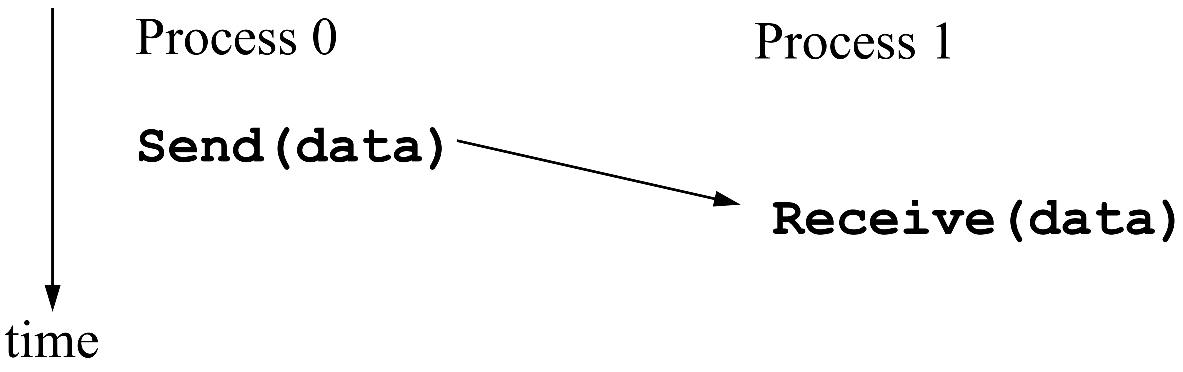
Inte

- MPI == ***Message-Passing Interface specification***
  - Extended message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or product
- Specified in C, C++, Fortran 77, F90
- Message Passing Interface (MPI) Forum
  - http://www.mpi-forum.org/
  - http://www.mpi-forum.org/docs/docs.html

- Two flavors for communication
  - Cooperative operations
  - One-sided operations

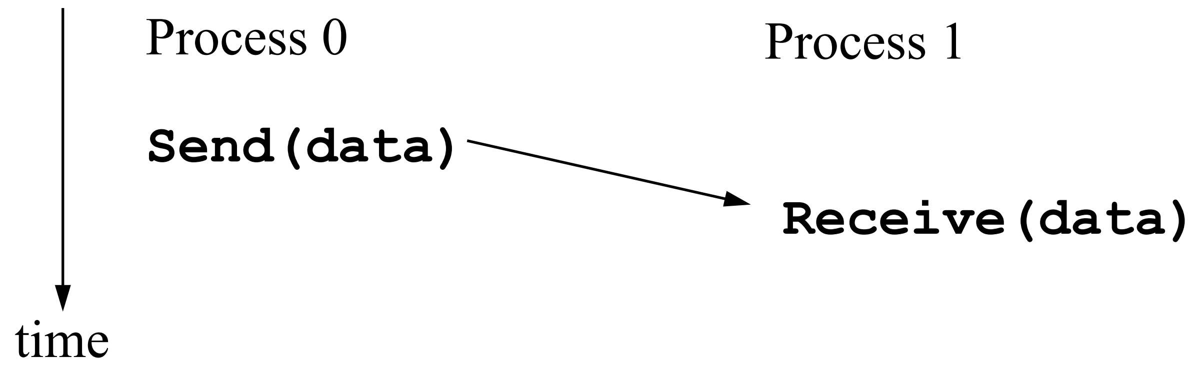
Cooperative Operations

Cooperative Operations



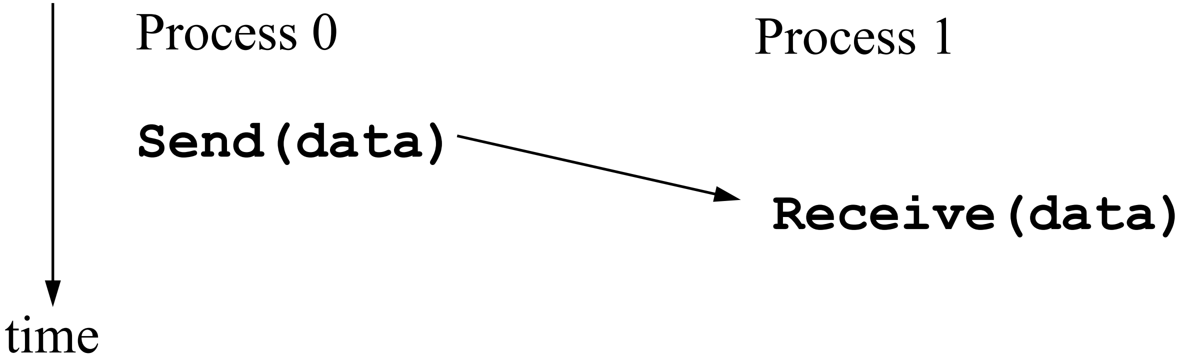
Cooperative Operations

Data is cooperatively exchanged in message-passing



Cooperative Operations

Data is cooperatively exchanged in message-passing
Explicitly sent by one process and received by another



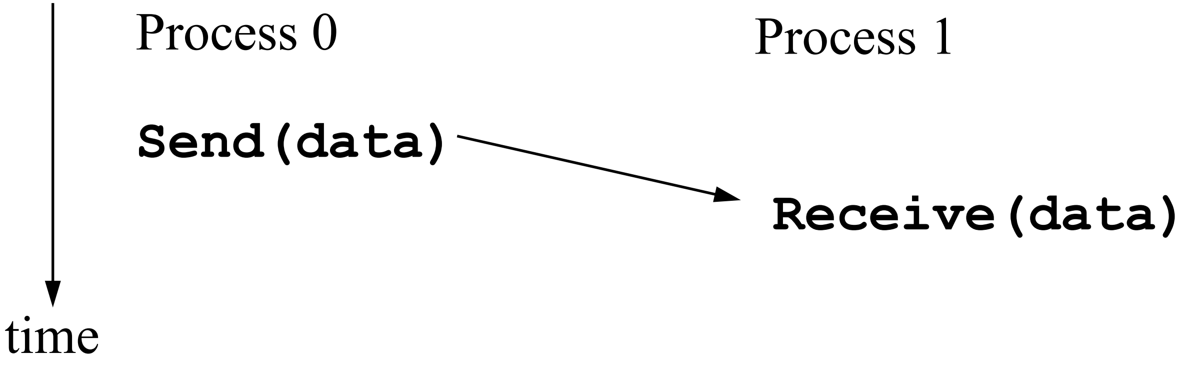
Cooperative Operations

Data is cooperatively exchanged in message-passing

Explicitly sent by one process and received by another

Advantage of local control of memory

- Change in the receiving process's memory made with receiver's explicit participation



Cooperative Operations

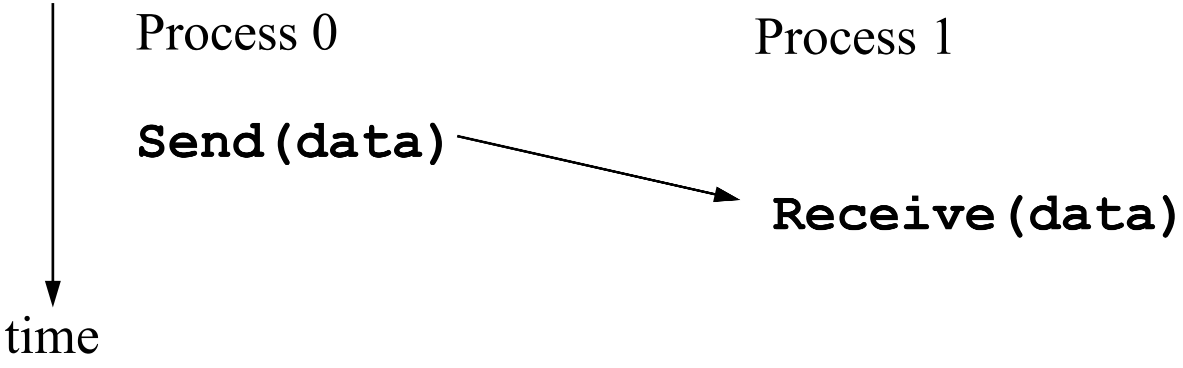
Data is cooperatively exchanged in message-passing

Explicitly sent by one process and received by another

Advantage of local control of memory

- Change in the receiving process's memory made with receiver's explicit participation

Communication and synchronization are combined



Cooperative Operations

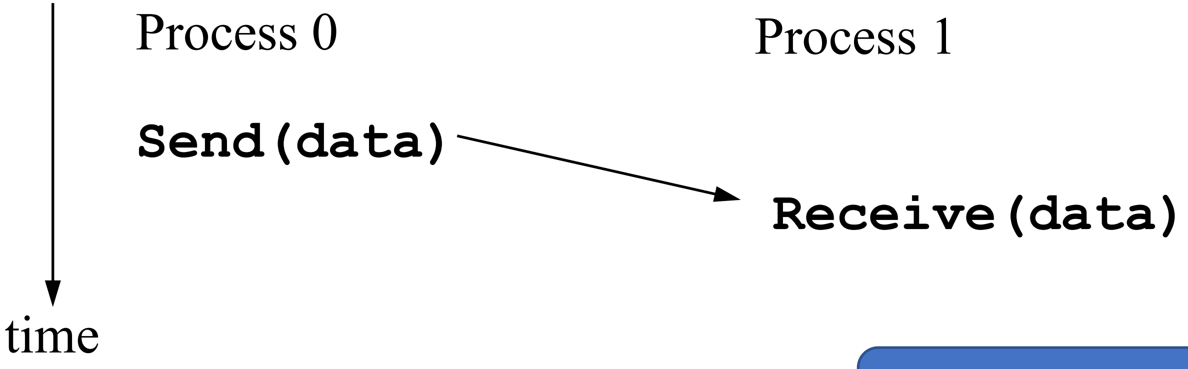
Data is cooperatively exchanged in message-passing

Explicitly sent by one process and received by another

Advantage of local control of memory

- Change in the receiving process's memory made with receiver's explicit participation

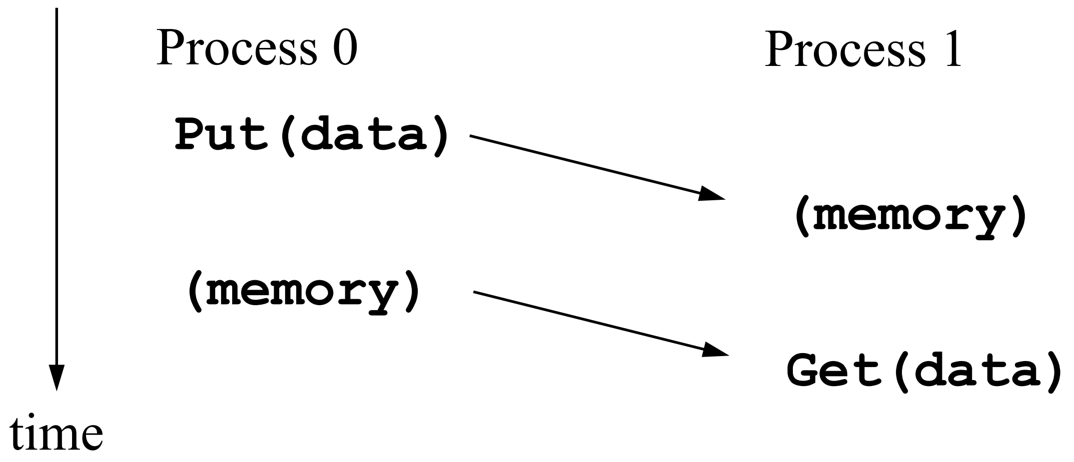
Communication and synchronization are combined



Familiar argument?

One-Sided Operations

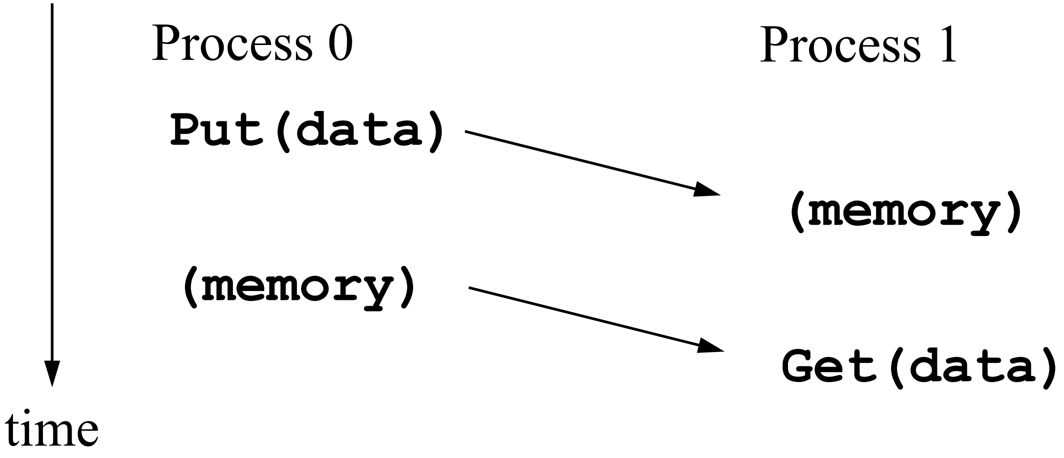
One-Sided Operations



One-Sided Operations

One-sided operations between processes

- Include remote memory reads and writes



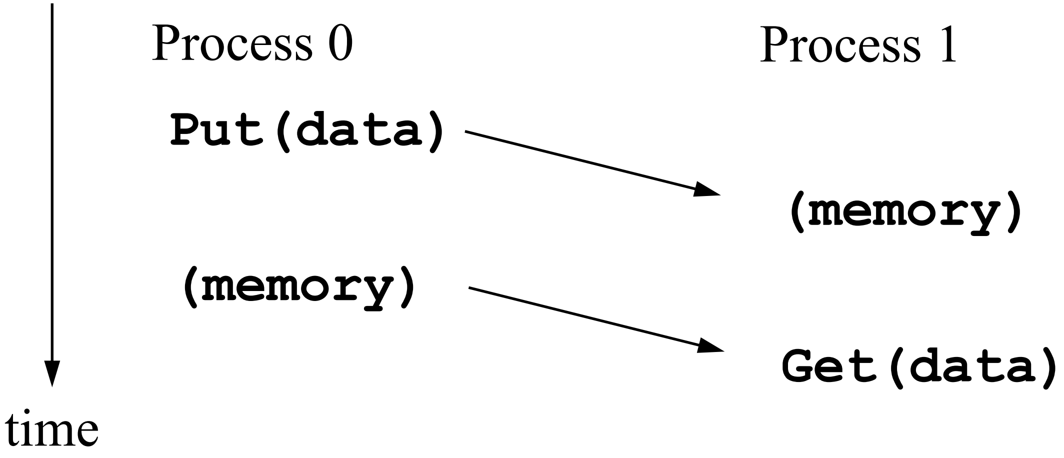
One-Sided Operations

One-sided operations between processes

- Include remote memory reads and writes

Only one process needs to explicitly participate

- There is still agreement implicit in the SPMD program



One-Sided Operations

One-sided operations between processes

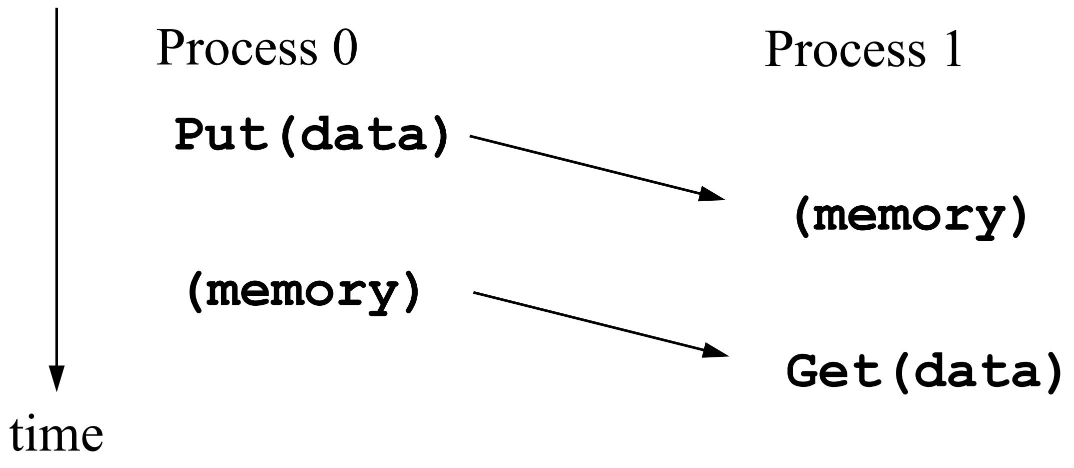
- Include remote memory reads and writes

Only one process needs to explicitly participate

- There is still agreement implicit in the SPMD program

Implication:

- Communication and synchronization are decoupled



One-Sided Operations

Are 1-sided operations better for performance?

One-sided operations between processes

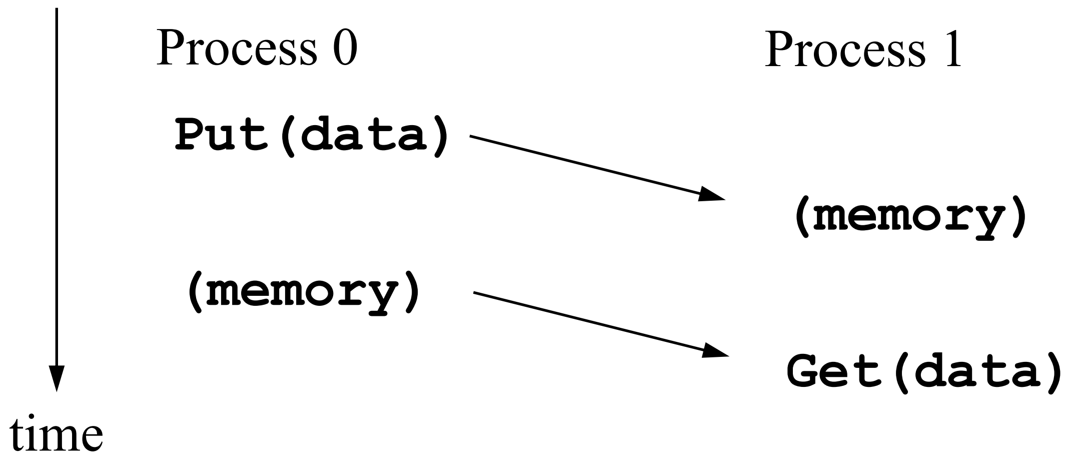
- Include remote memory reads and writes

Only one process needs to explicitly participate

- There is still agreement implicit in the SPMD program

Implication:

- Communication and synchronization are decoupled



A Simple MPI Program

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

MPI_Init

MPI_Init

Hardware resources allocated

- MPI-managed ones anyway…

MPI_Init

Hardware resources allocated

- MPI-managed ones anyway…

Start processes on different nodes

- Where does their executable program come from?

MPI_Init

- Hardware resources allocated
  - MPI-managed ones anyway…
- Start processes on different nodes
  - Where does their executable program come from?
- Give processes what they need to know
  - Wait…what do they need to know?

MPI_Init

- Hardware resources allocated
  - MPI-managed ones anyway…
- Start processes on different nodes
  - Where does their executable program come from?
- Give processes what they need to know
  - Wait…what do they need to know?
- Configure OS-level resources

MPI_Init

- Hardware resources allocated
  - MPI-managed ones anyway…
- Start processes on different nodes
  - Where does their executable program come from?
- Give processes what they need to know
  - Wait…what do they need to know?
- Configure OS-level resources
- Configure tools that are running with MPI

MPI_Init

Hardware resources allocated

- MPI-managed ones anyway…

Start processes on different nodes

- Where does their executable program come from?

Give processes what they need to know

- Wait…what do they need to know?

Configure OS-level resources

Configure tools that are running with MPI

…

MPI_Finalize

MPI_Finalize

Why do we need to finalize MPI?

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a “graceful” MPI exit?

- Can bad things happen otherwise?
- Suppose one process exits…

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a "graceful" MPI exit?

- Can bad things happen otherwise?
- Suppose one process exits…

How do resources get de-allocated?

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a “graceful” MPI exit?

- Can bad things happen otherwise?
- Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

MPI_Finalize

- Why do we need to finalize MPI?
- What is necessary for a “graceful” MPI exit?
  - Can bad things happen otherwise?
  - Suppose one process exits…
- How do resources get de-allocated?
- How to shut down communication?
- What type of exit protocol might be used?

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a "graceful" MPI exit?

- Can bad things happen otherwise?
- Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

What type of exit protocol might be used?

Executive Summary

- Undo all of init
- Be able to do it on success or failure exit

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a "graceful" MPI exit?

Can bad things happen otherwise?

Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

What type of exit protocol might be used?

Executive Summary

- Undo all of init
- Be able to do it on success or failure exit

- By default, an error causes all processes to abort

MPI_Finalize

Why do we need to finalize MPI?

What is necessary for a “graceful” MPI exit?

- Can bad things happen otherwise?
- Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

What type of exit protocol might be used?

Executive Summary

- Undo all of init
- Be able to do it on success or failure exit

- By default, an error causes all processes to abort
- The user can cause routines to return (with an error code)
  - In C++, exceptions are thrown (MPI-2)

MPI_Finalize

Executive Summary

- Undo all of init
- Be able to do it on success or failure exit

Why do we need to finalize MPI?

What is necessary for a "graceful" MPI exit?

Can bad things happen otherwise?

Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

What type of exit protocol might be used?

- By default, an error causes all processes to abort
- The user can cause routines to return (with an error code)
  - In C++, exceptions are thrown (MPI-2)
- A user can also write and install custom error handlers

MPI_Finalize

Executive Summary

- Undo all of init
- Be able to do it on success or failure exit

Why do we need to finalize MPI?

What is necessary for a “graceful” MPI exit?

Can bad things happen otherwise?

Suppose one process exits…

How do resources get de-allocated?

How to shut down communication?

What type of exit protocol might be used?

- By default, an error causes all processes to abort
- The user can cause routines to return (with an error code)
  - In C++, exceptions are thrown (MPI-2)
- A user can also write and install custom error handlers
- Libraries may handle errors differently from applications

Running MPI Programs

Running MPI Programs

MPI-1 does not specify how to run an MPI program

Running MPI Programs

MPI-1 does not specify how to run an MPI program

Starting an MPI program is dependent on implementation

- Scripts, program arguments, and/or environment variables

Running MPI Programs

MPI-1 does not specify how to run an MPI program

Starting an MPI program is dependent on implementation

- Scripts, program arguments, and/or environment variables

% mpirun -np <procs> a.out

- **For MPICH under Linux**

Running MPI Programs

MPI-1 does not specify how to run an MPI program

Starting an MPI program is dependent on implementation

- Scripts, program arguments, and/or environment variables

% mpirun -np <procs> a.out

- **For MPICH under Linux**

mpiexec <args>

- Recommended part of MPI-2, as a recommendation
- **mpiexec** for MPICH (distribution from ANL)
- **mpirun** for SGI's MPI

Finding Out About the Environment

Finding Out About the Environment

Two important questions that arise in message passing

- How many processes are being use in computation?
- Which one am I?

Finding Out About the Environment

Two important questions that arise in message passing

- How many processes are being use in computation?
- Which one am I?

MPI provides functions to answer these questions

- **MPI_Comm_size** reports the number of processes
- **MPI_Comm_rank** reports the rank
  - number between 0 and size-1
  - identifies the calling process

Hello World Revisited

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Hello World Revisited

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

- ❒ What does this program do?

Hello World Revisited

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

- What does this program do?

Comm?
"Communicator"

Basic Concepts

Processes can be collected into *groups*

Each message is sent in a *context*

- Must be received in the same context!

A group and context together form a *communicator*

A process is identified by its *rank*

- With respect to the group associated with a communicator

There is a default communicator MPI_COMM_WORLD

- Contains all initial processes

MPI Datatypes

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

An MPI *datatype* is recursively defined as:

- Predefined data type from the language
- A contiguous array of MPI datatypes
- A strided block of datatypes
- An indexed array of blocks of datatypes
- An arbitrary structure of datatypes

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

An MPI *datatype* is recursively defined as:

- Predefined data type from the language
- A contiguous array of MPI datatypes
- A strided block of datatypes
- An indexed array of blocks of datatypes
- An arbitrary structure of datatypes

There are MPI functions to construct custom datatypes

- Array of (int, float) pairs
- Row of a matrix stored columnwise

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

An MPI *datatype* is recursively defined as:

- Predefined data type from the language
- A contiguous array of MPI datatypes
- A strided block of datatypes
- An indexed array of blocks of datatypes
- An arbitrary structure of datatypes

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

An MPI *datatype* is recursively defined as:

Predefined data type from the language

A contiguous array of MPI datatypes

A strided block of datatypes

An indexed array of blocks of datatypes

An arbitrary structure of datatypes

- Enables heterogeneous communication
  - Support communication between processes on machines with different memory representations and lengths of elementary datatypes
  - MPI provides the representation translation if necessary

MPI Datatypes

Message data (sent or received) is described by a triple

address, count, datatype

An MPI *datatype* is recursively defined as:

Predefined data type from the language

A contiguous array of MPI datatypes

A strided block of datatypes

An indexed array of blocks of datatypes

An arbitrary structure of datatypes

- Enables heterogeneous communication
  - Support communication between processes on machines with different memory representations and lengths of elementary datatypes
  - MPI provides the representation translation if necessary
- Allows application-oriented layout of data in memory
  - Reduces memory-to-memory copies in implementation
  - Allows use of special hardware (scatter/gather)

MPI Tags

MPI Tags

Messages are sent with an accompanying user-defined integer *tag*

- Assist the receiving process in identifying the message

MPI Tags

- Messages are sent with an accompanying user-defined integer *tag*
  - Assist the receiving process in identifying the message
- Messages can be screened at receiving end by specifying specific tag
  - **MPI_ANY_TAG** matches any tag in a receive

MPI Tags

- Messages are sent with an accompanying user-defined integer *tag*
  - Assist the receiving process in identifying the message
- Messages can be screened at receiving end by specifying specific tag
  - **MPI_ANY_TAG** matches any tag in a receive
- Tags are sometimes called "message types"
  - MPI calls them "tags" to avoid confusion with datatypes

MPI Basic (Blocking) Send

`MPI_SEND (start, count, datatype, dest, tag, comm)`

- The message buffer is described by:
  - **`start`**, **`count`**, **`datatype`**
- The target process is specified by **`dest`**
  - Rank of the target process in the communicator specified by **`comm`**
- Process blocks until:
  - Data has been delivered to the system
  - Buffer can then be reused
- Message may not have been received by target process!

MPI with Only Six Functions

MPI with Only Six Functions

Many parallel programs can be written using:

MPI_INIT()

MPI_FINALIZE()

MPI_COMM_SIZE()

MPI_COMM_RANK()

MPI_SEND()

MPI_RECV()

MPI with Only Six Functions

Many parallel programs can be written using:

MPI_INIT()
MPI_FINALIZE()
MPI_COMM_SIZE()
MPI_COMM_RANK()
MPI_SEND()
MPI_RECV()

Why have any other APIs (e.g. broadcast, reduce, etc.)?

MPI with Only Six Functions

Many parallel programs can be written using:

- **MPI_INIT()**
- **MPI_FINALIZE()**
- **MPI_COMM_SIZE()**
- **MPI_COMM_RANK()**
- **MPI_SEND()**
- **MPI_RECV()**

Why have any other APIs (e.g. broadcast, reduce, etc.)?

Point-to-point (send/recv) isn't always the most efficient...

- Add more support for communication

Excerpt: Barnes-Hut

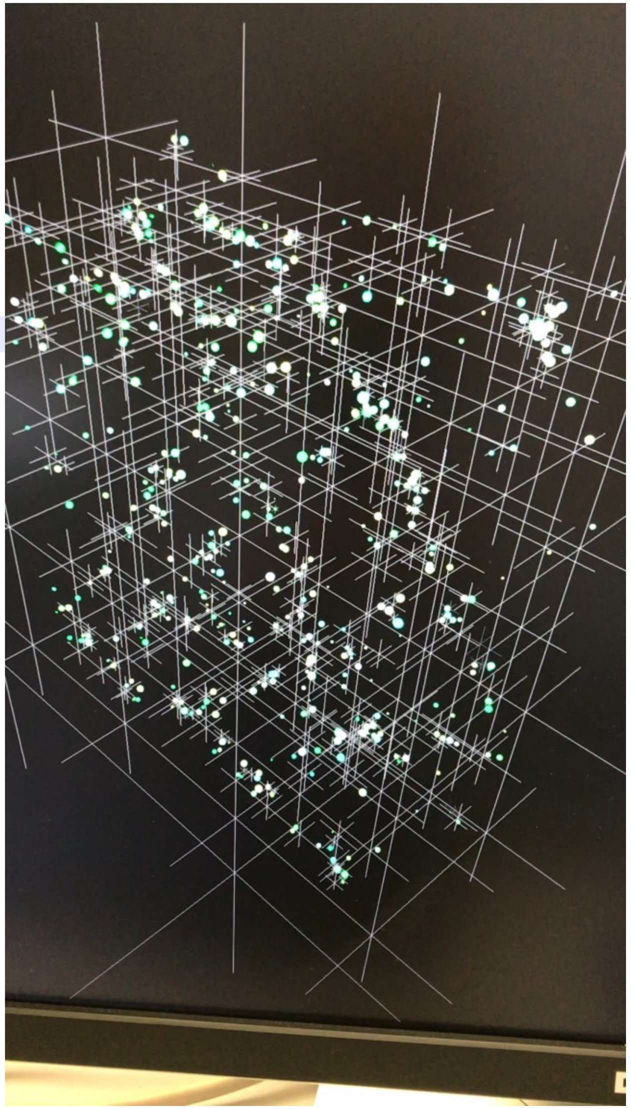
```
int ctr=nLocalOriginal;
int offset=nLocalOriginal-nLocal;
for(i=0;i<worldSize;i++){
if(i==rank){
    MPI_Bcast(s_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,i,MPI_COMM_WORLD);
} else {
    MPI_Bcast(l_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,i,MPI_COMM_WORLD);
    for(k=0;k<l_particles[0];k++, ctr++){
    if(l_particles[MASS(k)]<0){
        offset++;
        _nparticles--;
    } else {
        s_particles[PX(ctr)]=l_particles[PX(k)];
        s_particles[PY(ctr)]=l_particles[PY(k)];
        s_particles[PZ(ctr)]=l_particles[PZ(k)];
        s_particles[MASS(ctr)]=l_particles[MASS(k)];
        indexes[ctr-offset]=ctr;
    }
}
```

Excerpt: Barnes-Hut

```
int ctr=nLocalOriginal;
int offset=nLocalOriginal-nLocal;
for(i=0;i<worldSize;i++){
if(i==rank){
    MPI_Bcast(s_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,i,MPI_COMM_WORLD);
} else {
    MPI_Bcast(l_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,i,MPI_COMM_WORLD);
    for(k=0;k<l_particles[0];k++, ctr++){
    if(l_particles[MASS(k)]<0){
        offset++;
        _nparticles--;
    } else {
        s_particles[PX(ctr)]=l_particles[PX(k)];
        s_particles[PY(ctr)]=l_particles[PY(k)];
        s_particles[PZ(ctr)]=l_particles[PZ(k)];
        s_particles[MASS(ctr)]=l_particles[MASS(k)];
        indexes[ctr-offset]=ctr;
    }
}
```

Excerpt: Barnes-Hut

```
int ctr=nLocalOriginal;
int offset=nLocalOriginal-nLocal;
for(i=0;i<worldSize;i++){
if(i==rank){
    MPI_Bcast(s_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,
} else {
    MPI_Bcast(l_particles,N_POS_ELEMS*nLocalMax+1,MPI_DOUBLE,
    for(k=0;k<l_particles[0];k++, ctr++){
    if(l_particles[MASS(k)]<0){
        offset++;
        _nparticles--;
    } else {
        s_particles[PX(ctr)]=l_particles[PX(k)];
        s_particles[PY(ctr)]=l_particles[PY(k)];
        s_particles[PZ(ctr)]=l_particles[PZ(k)];
        s_particles[MASS(ctr)]=l_particles[MASS(k)];
        indexes[ctr-offset]=ctr;
    }
}
```



To use or not use MPI?

To use or not use MPI?

- USE
  - You need a portable parallel program
  - You are writing a parallel library
  - You have irregular or dynamic data relationships
  - You care about performance

To use or not use MPI?

- USE
  - You need a portable parallel program
  - You are writing a parallel library
  - You have irregular or dynamic data relationships
  - You care about performance
- NOT USE
  - You don’t need parallelism at all
  - You can use libraries (which may be written in MPI) or other tools
  - You can use multi-threading in a concurrent environment
    - You don’t need extreme scale