Is Transactional Programming Actually Easier?

Christopher J. Rossbach,
Owen S. Hofmann,
Emmett Witchel
UT Austin

TM Research Mantra

- We need better parallel programming tools
  - CMP ubiquity
  - (Concurrent programming == programming w/locks)
  - Locks are difficult
- Transactional memory is “promising”:
  - No deadlock, livelock, etc.
  - Optimistic → likely more scalable
- **Therefore**:
  - Transactional Memory is ***easier*** than locks
  - All TM papers should be published

Is TM *really* easier than locks?

- Programmers ***still must write critical sections***
- Realizable TM will have *new* issues
  - HTM overflow
  - STM performance
  - Trading one set of difficult issues for another?
- Ease-of-use is a critical motivator for TM research

It's important to know the answer to this question

How can we answer this question?

Step 1: Get some programmers

(preferrab

Step 2: hav

program v

Step 3: Ask

Step 4: Eva

This talk:

- TM vs. locks user study
- UT Austin OS undergrads
- same program using
  - locks (fine/coarse)
  - monitors
  - transactional memory

Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion

The programming problem

- **sync-gallery**: a rogue's gallery of synchronization
- Metaphor → shooting gallery (welcome to UT)
- Rogues → shoot paint-balls in lanes
  - Each rogue has a unique color
- Shooting → target takes rogue's color
- Cleaners → change targets back to white
- Rogues/cleaners must synchronize
  - **maintain 4 invariants**

Sync-gallery invariants

- Only one shooter per lane **(Uh, hello, dangerous?!)**
- Don't shoot colored lanes **(no fun)**
- Clean only when all lanes shot **(be lazy)**
- Only one cleaner thread

Task: “single-lane rogue”

```
Rogue() {
  while(true) {
    Lane lane = randomLane();
    if(lane.getColor() == WHITE)
      lane.shoot();
    if(allLanesShot())
      clean();
  }
}
```

beginTransaction()

globalLock.lock()

lane.lock()

lane.unlock()

lockAllLanes() ???

endTransaction()

globalLock.unlock()

Coarse-grain locking

Fine-grain locking

Transactions

Invariants:

- One shooter per lane
- Don’t shoot colored lanes
- One cleaner thread
- Clean only when all lanes shot

Variation: “two-lane rogue”

```
Rogue() {
  while(true) {
    Lane a = randomLane();
    Lane b = randomLane();
    if(a.getColor() == WHITE &&
       b.getColor() == WHITE) {
      a.shoot();
      b.shoot();
    }
    if(allLanesShot())
      clean();
}}
```

globalLock.lock()

a.lock();
b.lock();

Requires lock-ordering!

lockAllLanes() ???

globalLock.unlock()

Coarse-grain locking

Fine-grain locking

Invariants:

- One shooter per lane
- Don’t shoot colored lanes
- One cleaner thread
- Clean only when all lanes shot

Variation 2: “cleaner rogues”

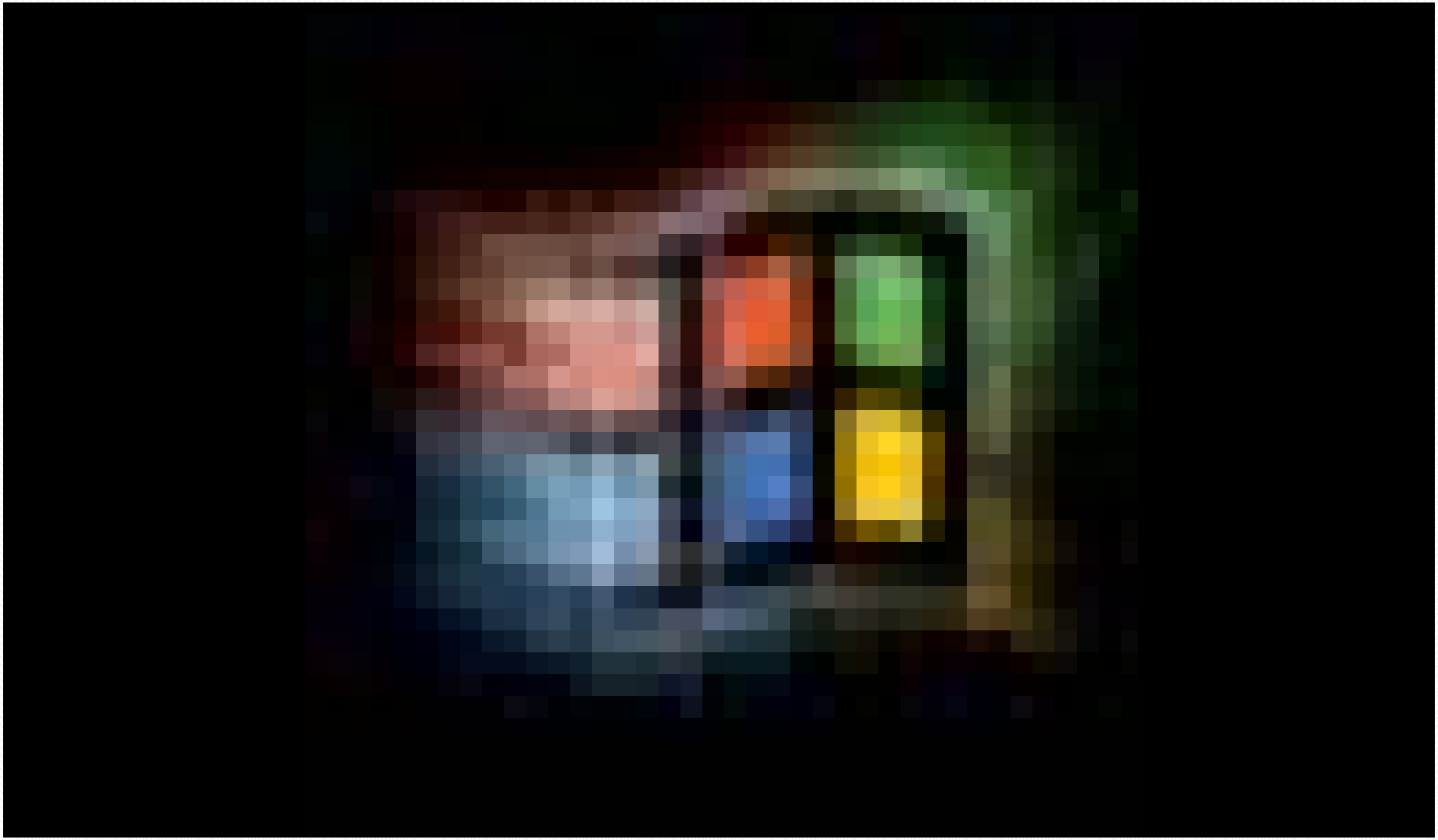
```
Rogue() {
  while(true)
    Lane lane = randomLane();
    if(lane.getColor() == WHITE)  if(allLanesShot())
      lane.shoot();                   lanesFull.signal();
} }
Cleaner() {                      while(!allLanesShot()
  while(true) {                    lanesFull.await()
    if(allLanesShot())
      clean();
} }
```

(still need other locks!)

Invariants:

- One shooter per lane
- Don’t shoot colored lanes
- One cleaner thread
- Clean only when all lanes shot

Sync-gallery in action



Synchronization Cross-product

	Coarse	Fine	TM
Single-lane	Coarse	Fine	TM
Two-lane	Coarse2	Fine2	TM2
Cleaner	CoarseCleaner	FineCleaner	TMCleaner

9 different Rogue implementations

Outline

- Motivation
- Programming Problem
- User Study Methodology
  - TM Support
  - Survey details
- Results
- Conclusion

TM Support

- Year 1: DSTM2 **[Herlihy 06]**
- Year 2: JDASTM **[Ramadan 09]**
- Library, not language support
  - No atomic blocks
  - Different concrete syntax
  - Read/write barriers

DSTM2 concrete syntax

```
Callable c = new Callable<Void> {
   public Void call() {
      GalleryLane l = randomLane();
      if(l.color() == WHITE))
         l.shoot(myColor);
      return null;
   }
}
Thread.doIt(c);
```

JDASTM concrete syntax

```
Transaction tx = new Transaction(id);
boolean done = false;
while(!done) {
   try {
      tx.BeginTransaction();
      GalleryLane l = randomLane();
      if(l.color() == WHITE))
         l.TM_shoot(myColor);
      done = tx.CommitTransaction();
   } catch(AbortException e) {
      tx.AbortTransaction();
      done = false;
   }}
```

Undergrads: the ideal TM user-base

- TM added to undergrad OS curriculum
- Survey students
- Analyze programming mistakes
- TM's benchmark for success
  - ***Easier to use than fine grain locks or conditions***

Survey

- Measure previous exposure
  - Used locks/TM before, etc
- Track design/code/debug time
- Rank primitives according along several axes:
  - Ease of reasoning about
  - Ease of coding/debugging
  - Ease of understanding others' code

http://www.cs.utexas.edu/~witchel/tx/sync-gallery-survey.html

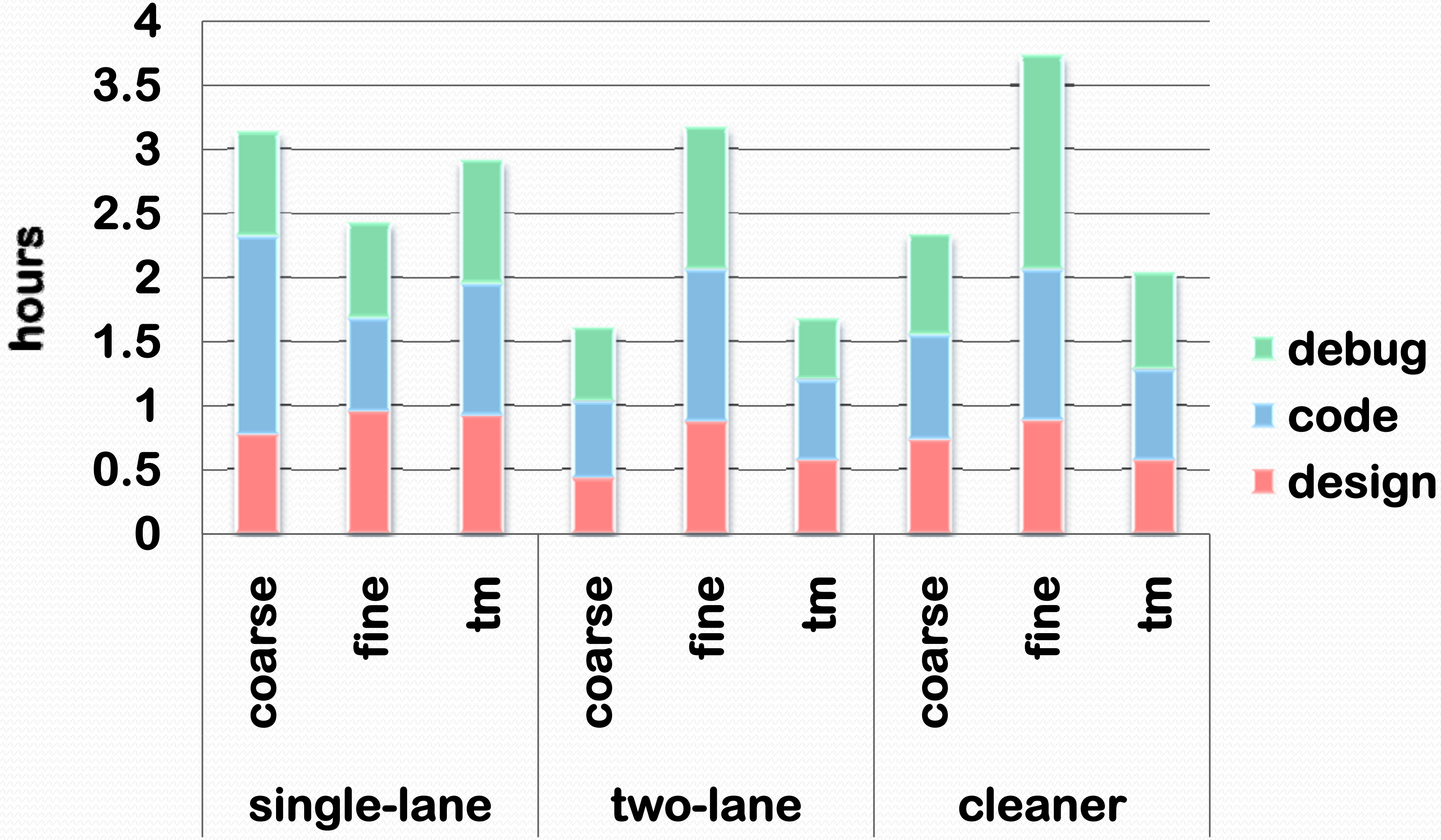
Data collection

- Surveyed 4 sections of OS students
  - 2 sections x 2 semesters
  - **147** students
  - **1323** rogue implementations
- Defect Analysis
  - Examined year 2 (**604** implementations)
  - Automated testing using condor

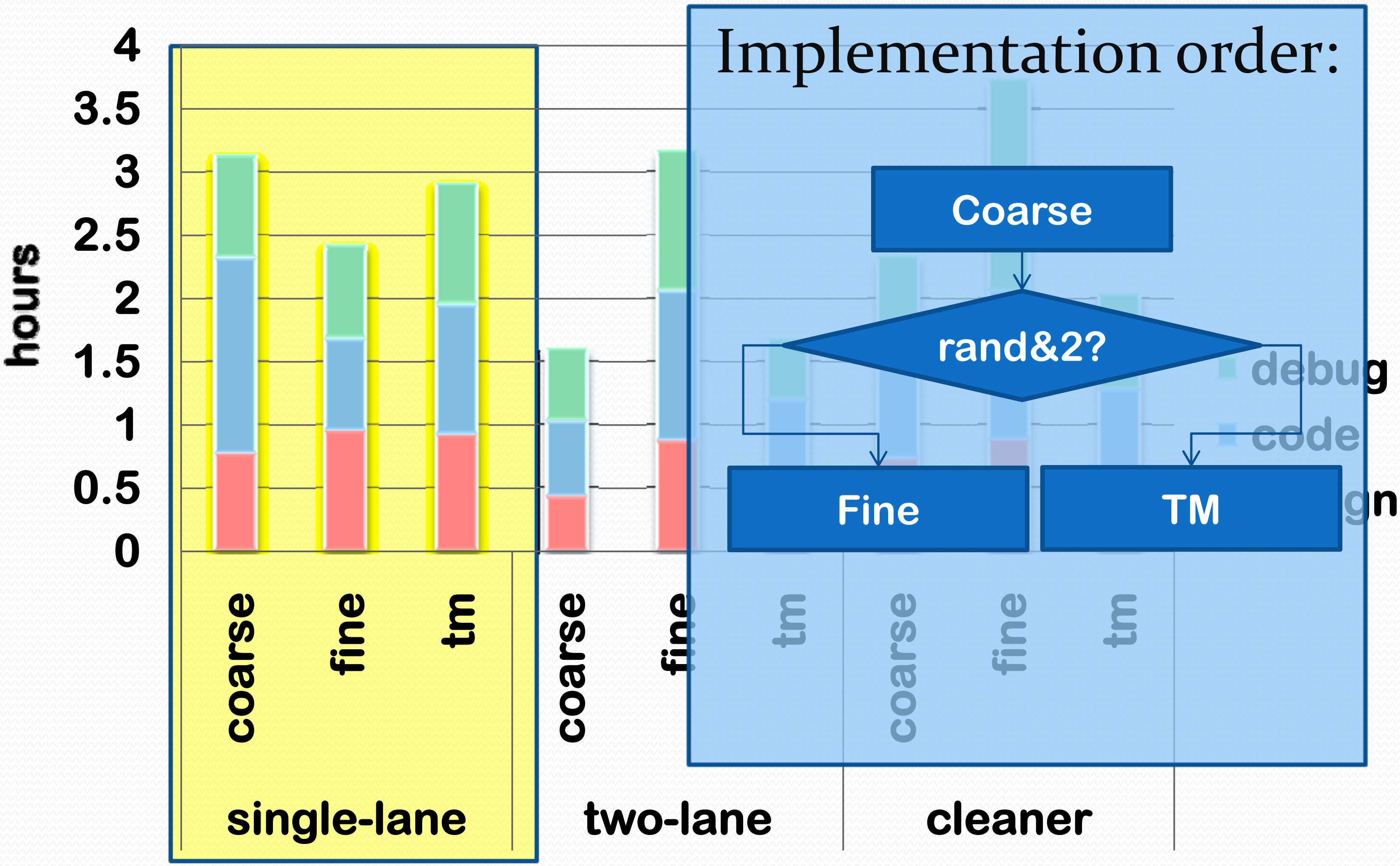
Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion

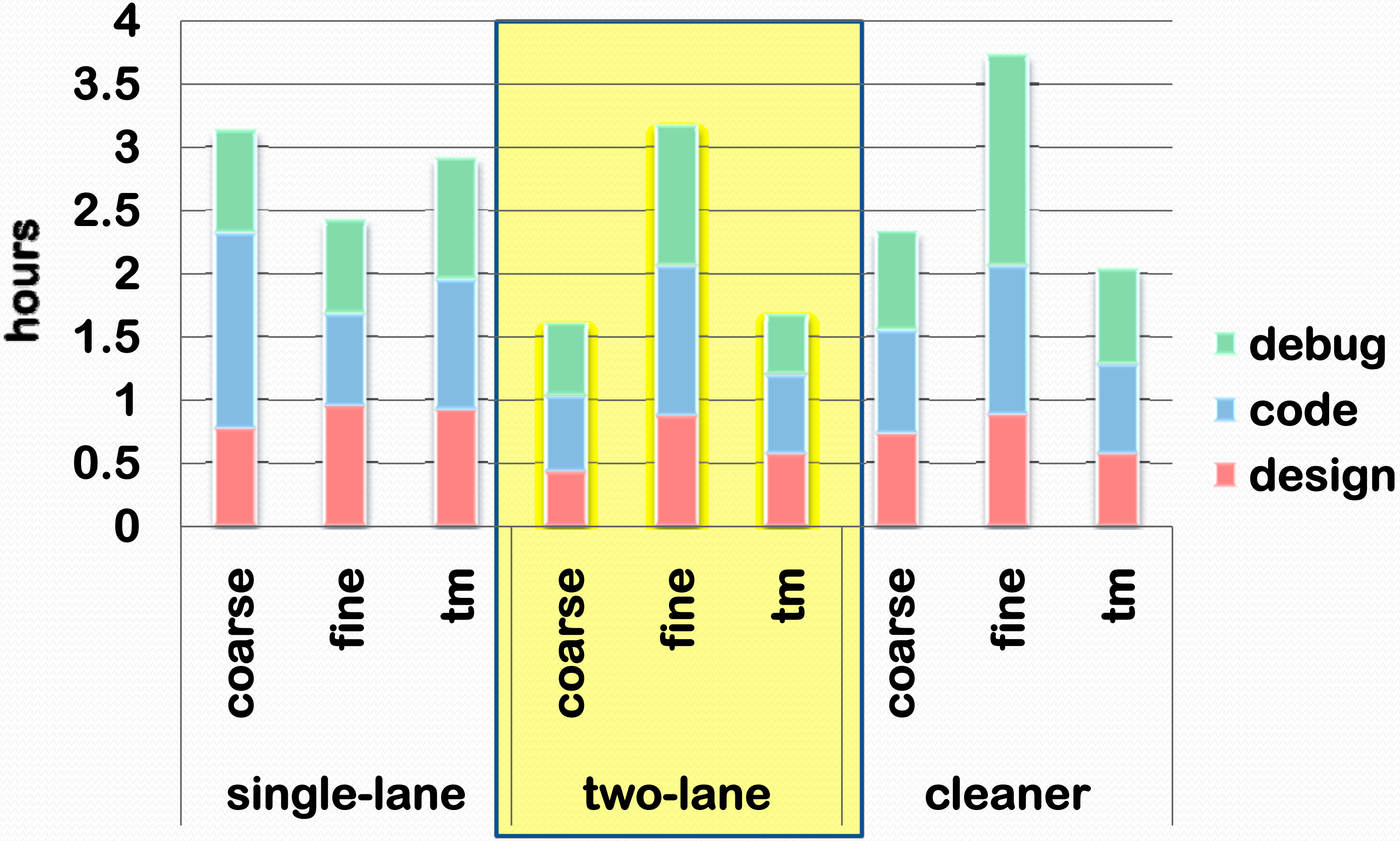
Development Effort



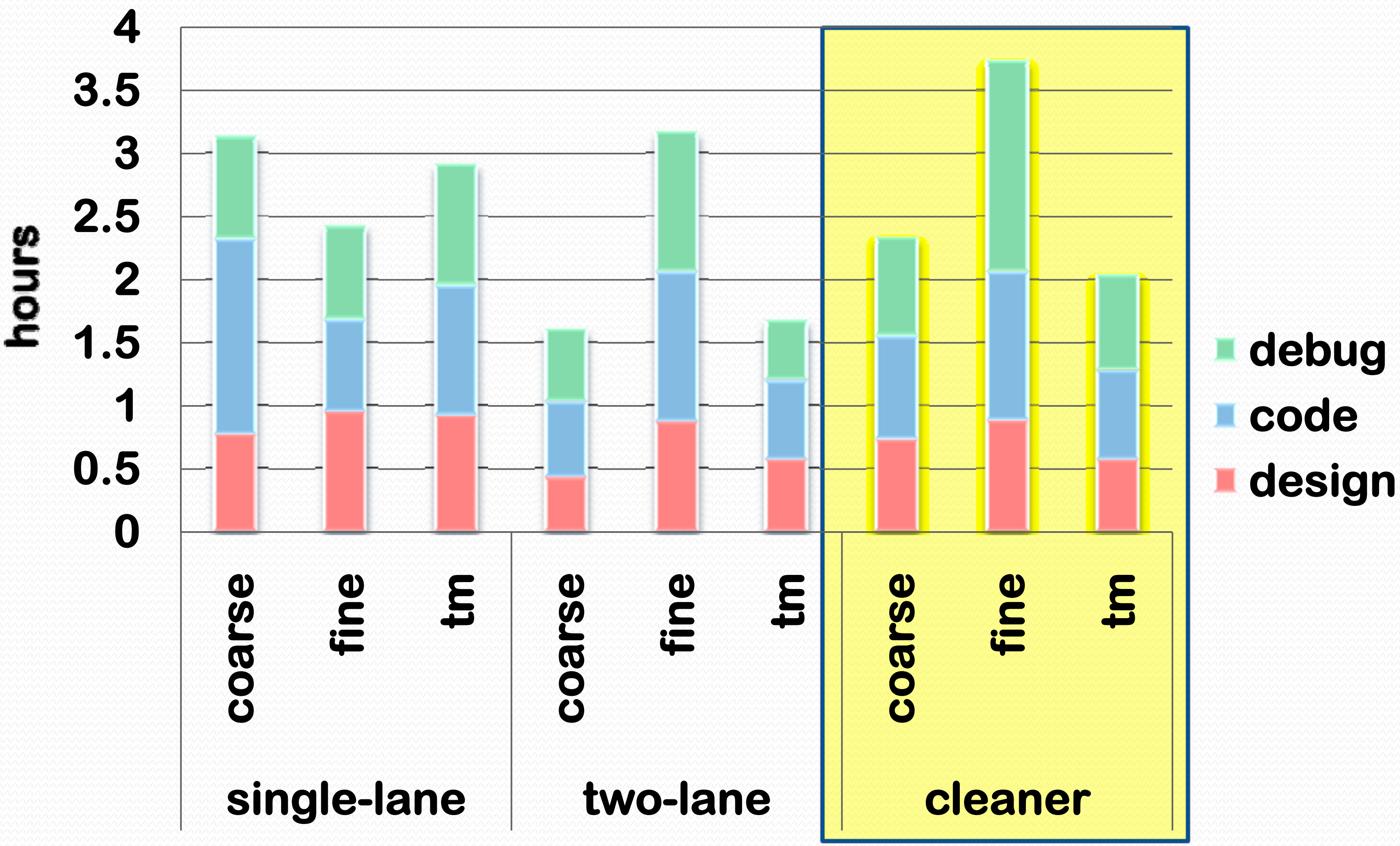
Development Effort



Development Effort



Development Effort



Qualitative preferences

Best Syntax

Ranking	1	2	3	4
Coarse	**62%**	30%	1%	4%
Fine	6%	21%	**45%**	**40%**
TM	26%	**32%**	19%	21%
Conditions	6%	21%	29%	**40%**

Easiest to Think about

Ranking	1	2	3	4
Coarse	**81%**	14%	1%	3%
Fine	1%	**38%**	30%	29%
TM	16%	**32%**	30%	21%
Conditions	4%	14%	**40%**	**40%**

(Year 2)

Qualitative preferences

Best Syntax

Coarse > (TM|Coarse) > Fine > (Fine|Conditions)

Ranking	1	2	3	4
Coarse	[illegible]	[illegible]	[illegible]	4%
Fine	6%	[illegible]	[illegible]	40%
TM	26%	32%	19%	21%
Conditions	6%	21%	29%	40%

Easiest to Think about

Coarse > (Fine|TM) > Conditions

Ranking	1	2	3	4
Coarse	[illegible]	[illegible]	[illegible]	[illegible]
Fine	1%	38%	30%	29%
TM	16%	32%	30%	21%
Conditions	4%	14%	40%	40%

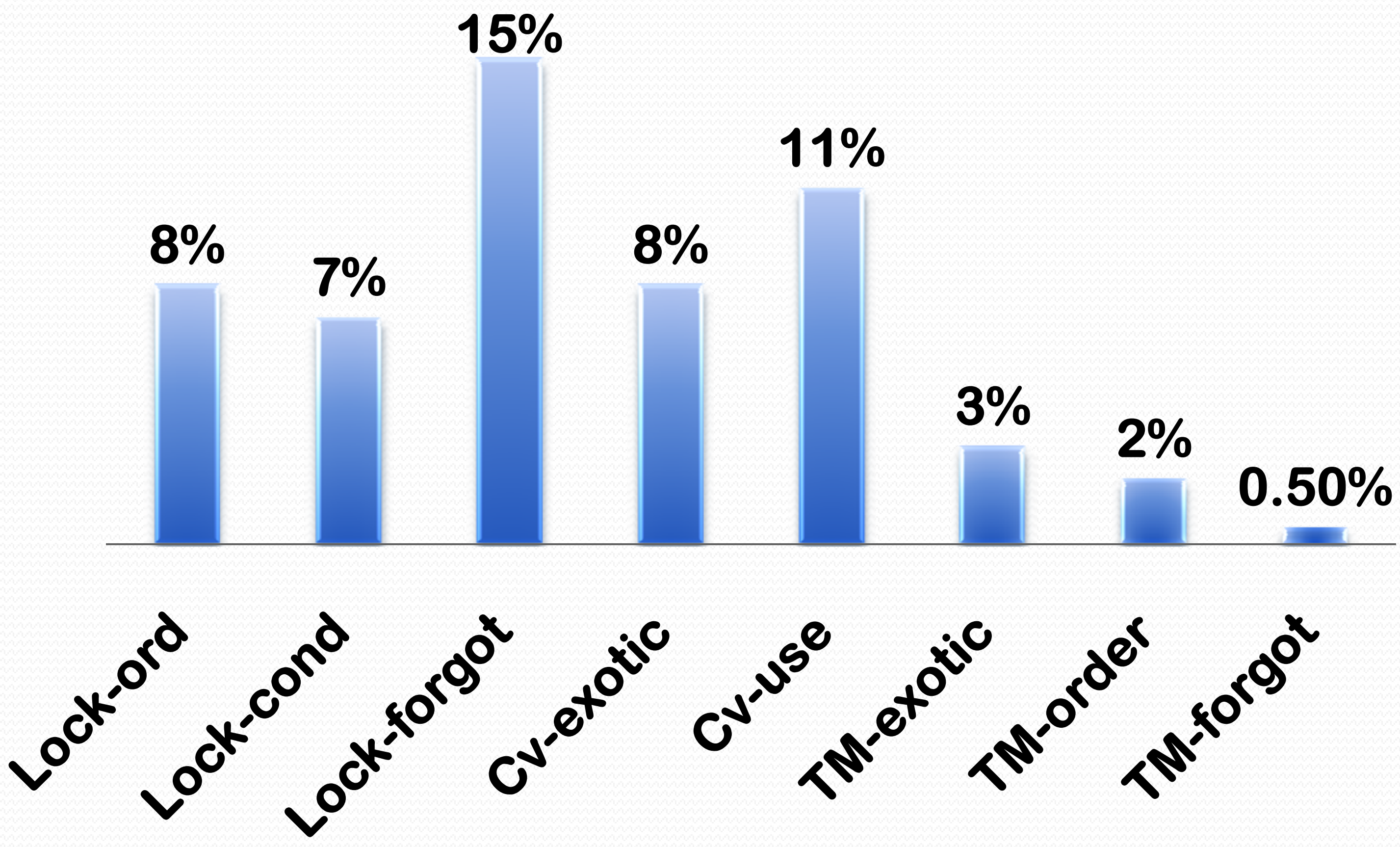
(Year 2)

Analyzing Programming Errors

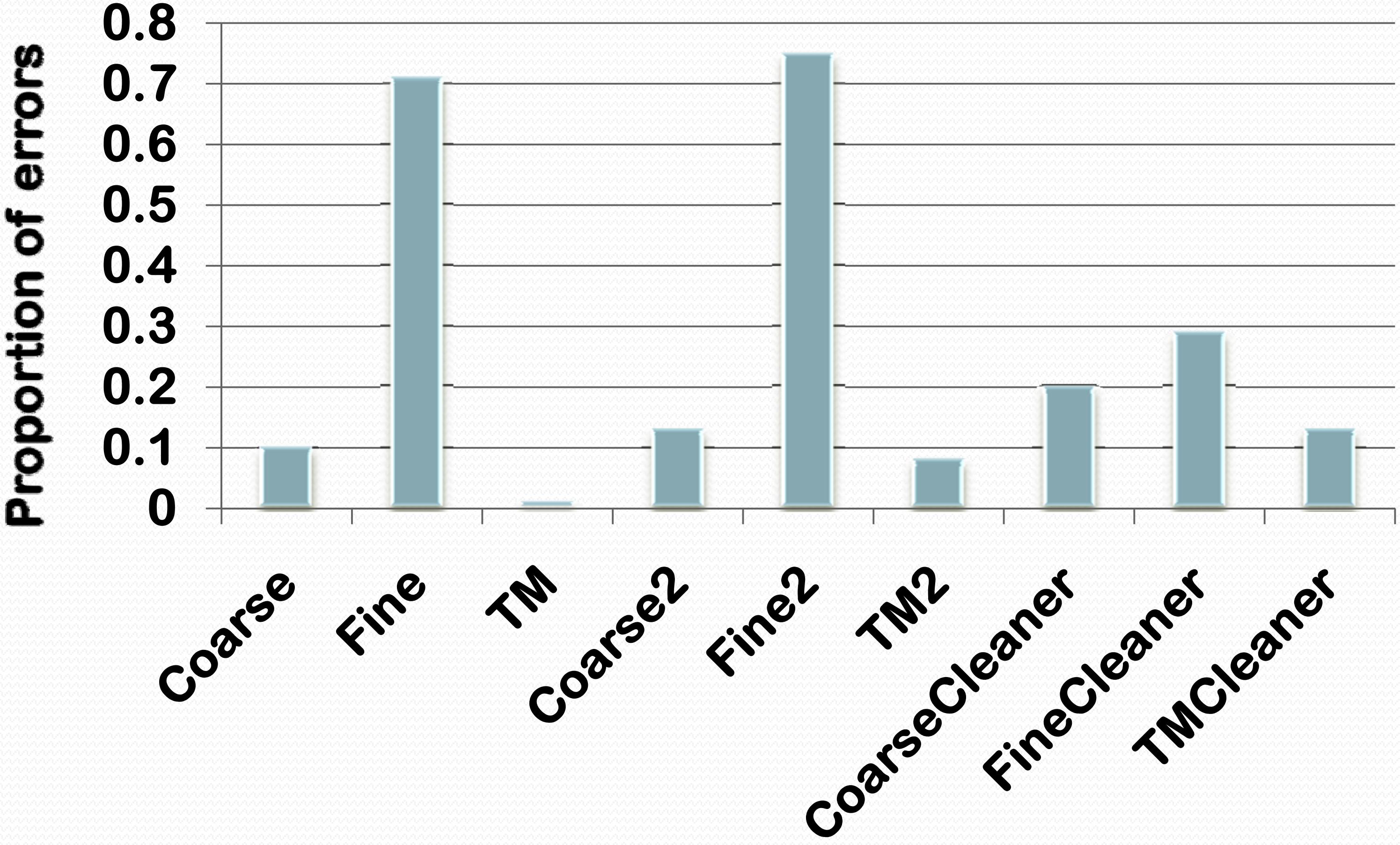
Error taxonomy: 8 classes

- **Lock-ord:** lock ordering
- **Lock-cond:** checking condition outside critsec
- **Lock-forgot:** forgotten Synchronization
- **Cv-exotic:** exotic condition variable usage
- **Cv-use:** condition variable errors
- **TM-exotic:** TM primitive misuse
- **TM-forgot:** Forgotten TM synchronization
- ***TM-order:*** *Ordering in TM*

Error Rates by Defect Type



Overall Error Rates



Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion

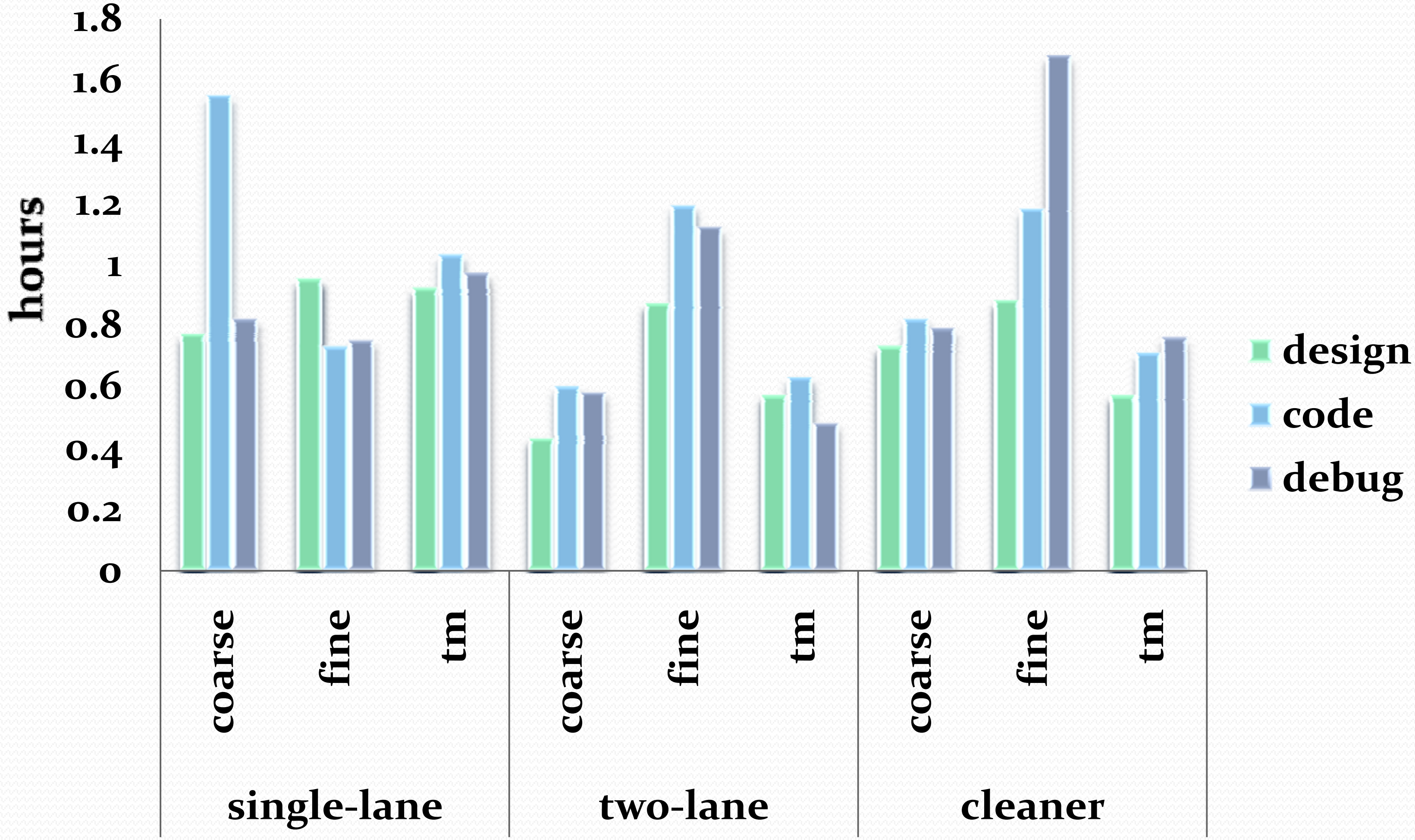
Conclusion

- General qualitative ranking:
  1. Coarse-grain locks (easiest)
  2. TM
  3. Fine-grain locks/conditions (hardest)
- Error rates overwhelmingly in favor of TM
- TM may ***actually be easier…***

P-values

		Coarse	Fine	TM	Coarse2	Fine2	TM2	CoarseCleaner	FineCleaner	TMCleaner
Coarse	Y1	1.00	**0.03**	**0.02**	1.00	**0.02**	1.00	0.95	0.47	0.73
	Y2	1.00	0.33	0.12	1.00	0.38	1.00	1.00	0.18	1.00
Fine	Y1	0.97	1.00	0.33	1.00	0.24	1.00	1.00	0.97	0.88
	Y2	0.68	1.00	0.58	1.00	0.51	1.00	1.00	0.40	1.00
TM	Y1	0.98	0.68	1.00	1.00	0.13	1.00	1.00	0.98	0.92
	Y2	0.88	0.43	1.00	1.00	0.68	1.00	1.00	0.41	1.00
Coarse2	Y1	**<0.01**	**<0.01**	**<0.01**	1.00	**<0.01**	**<0.01**	**<0.01**	**<0.01**	**<0.01**
	Y2	**<0.01**	**<0.01**	**<0.01**	1.00	**<0.01**	0.45	**<0.01**	**<0.01**	**<0.01**
Fine2	Y1	0.98	0.77	0.87	1.00	1.00	1.00	1.00	1.00	0.98
	Y2	0.62	0.49	0.32	1.00	1.00	1.00	0.99	0.59	1.00
TM2	Y1	**<0.01**	**<0.01**	**<0.01**	0.99	**<0.01**	1.00	**0.04**	**<0.01**	**<0.01**
	Y2	**<0.01**	**<0.01**	**<0.01**	0.55	**<0.01**	1.00	**<0.01**	**<0.01**	**<0.01**
CoarseCleaner	Y1	0.05	**<0.01**	**<0.01**	1.00	**<0.01**	0.96	1.00	**<0.01**	0.08
	Y2	**<0.01**	**<0.01**	**<0.01**	1.00	**<0.01**	1.00	1.00	**<0.01**	0.96
FineCleaner	Y1	0.53	**0.03**	**0.02**	1.00	**<0.01**	1.00	0.99	1.00	0.46
	Y2	0.83	0.60	0.59	1.00	0.42	1.00	1.00	1.00	1.00
TMCleaner	Y1	0.28	0.12	0.08	1.00	**0.03**	1.00	0.92	0.55	1.00
	Y2	**<0.01**	**<0.01**	**<0.01**	0.99	**<0.01**	1.00	**0.04**	**<0.01**	1.00

Development Effort



Synchronization Potpourri

Rogue	Synchronization	R/C Threads	Additional Requirements
Coarse	Global lock	not distinct	
Coarse2	Global lock	not distinct	Shoot at 2 lanes
CoarseCleaner	Global lock, conditions	Distinct	wait/notify
Fine	Per-lane locks	not distinct	
Fine2	Per-lane locks	not distinct	Shoot at 2 lanes
FineCleaner	Per-lane locks, conditions	Distinct	wait/notify
TM	TM	not distinct	
TM2	TM	not distinct	Shoot at 2 lanes
TMCleaner	TM	distinct	