Therac-25

Emmett Witchel

CS380L

Therac-25 takeaways

- Complex systems fail for complex reasons
- Users are often the experts on how the deployed system
  - Not the designers
  - Give the users enough information to help the designers (malfunction 54)
  - Make interaction explicit (mail errors, iOS app approval)
- If you want to build safety critical infrastructure…
  - Build and maintain an OS separately!
- If you want safety properties, specify them, at least informally
- Improbable cases can be safety critical!
  - Non-deterministic multi-threading is really hard to get right

Therac-25:

What is it? What happened?

- medical linear accelerator: treat cancer, remove tumors
- AECL + CGR collaborate in early 70s to build Therac-6 and -20
- 1976: AECL develops “double pass” tech. enabling Therac-25

❑ Shallow tissue treated with accelerated electrons

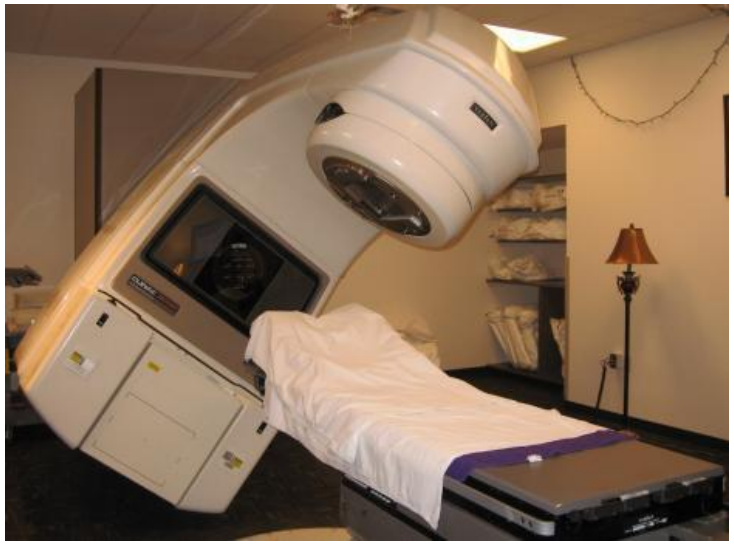
- Scanning magnets placed in the way of the beam; the spread of the beam (and thus power) controlled by magnetic fields

❑ Deeper tissue treated with X-ray photons

- X-ray beam flattened by a device in the machine to direct the appropriate intensity to the patient.

- Various horrifying accidents between 1985-87

Do not lay down here.



Anatomy of the accidents

At Texas facility

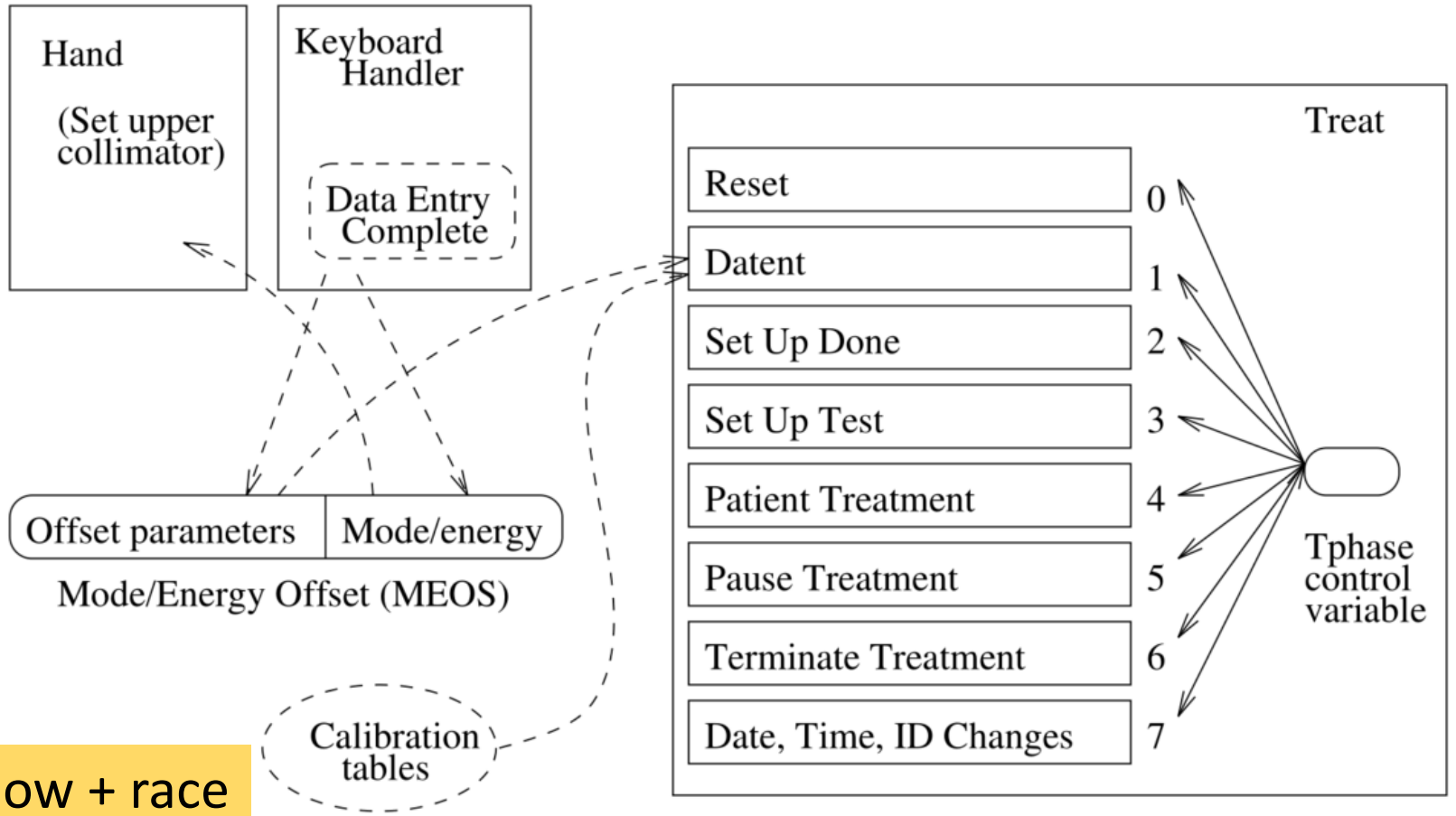
- Operator selects x-rays by mistake
- …used cursor keys to change to electrons
- Machine tripped with “Malfunction 54”
  - – Documentation explains this is “dose input 2” error
- Operator sees “beam ready” proceeds; go to 1

At Washington facility

- Operator puts table in field-light position to check alignment
- Operator sets machine but forgets to remove film
- Operator turns beam on, machine says no dose (+fleeting message)
- Operator proceeds from pause; After another pause, operator reenters room

What were the root causes?

Therac tasks & subroutines



Overflow + race

Racey

Pseudo-code

```
Datent {
  if(mode/energy specified) {
    calculate table index
    do {
      fetch parameter
      output parameter
      point to next parameter
    } until (all parameters set)
    call Magnet
    if(mode/energy changed)
      return
  }
  if(data-entry-complete) Tphase = 3
  else if(reset-command) Tphase = 0
}
```

```
Magnet {
  Set bending magnet flag
  do {
    set next magnet
    call Ptime
    if(mode/energy changed)
      exit
  } until (all magnets set)
}
```

```
Ptime {
  do {
    if (bending magnet flag)
      if (editing)
        if (mode/energy changed) exit
  } until hystereis delay expired
  clear bending magnet flag
}
```

Pseudo-code

```
Datent {
  if(mode/energy specified) {
    calculate table index
    do {
      fetch parameter
      output parameter
      point to next parameter
    } until (all p
    call Magnet
    if(mode/energy
      return
  }
  if(data-entry-complete) Tphase = 3
  else if(reset-command) Tphase = 0
}
```

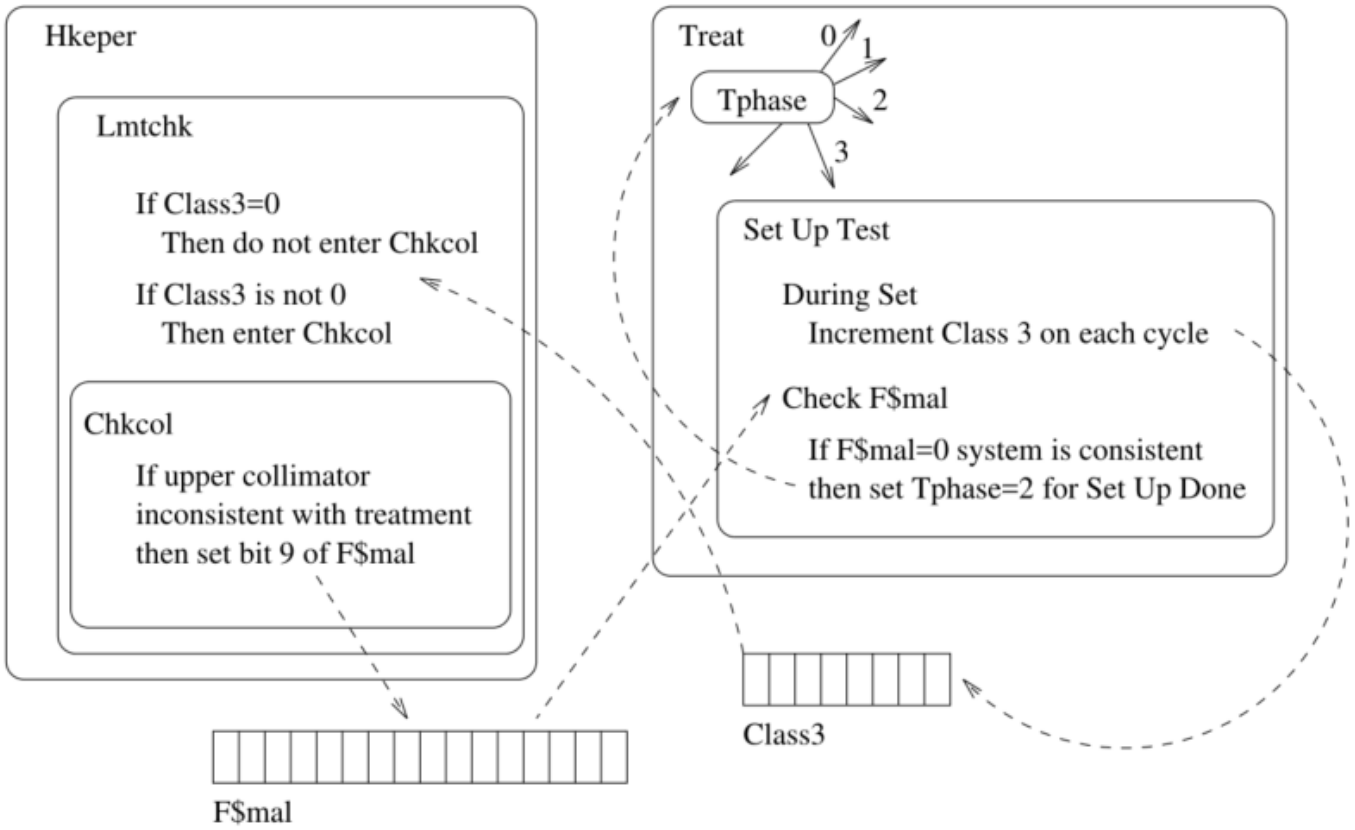
Only checks whether cursor has been to command line, not whether it's still there

```
Magnet {
  Set bending mag
  do {
    set next magn
    call Ptime
    if(mode/energ
      exit
  } until (all magnets set)
}
```

Called multiple times, but bending magnet cleared after first call, so changes after first Ptime *not* recognized: Shown on screen, but parameters not changed

```
Ptime {
  do {
    if (bending magnet flag)
      if (editing)
        if (mode/energy changed) exit
  } until hystereis delay expired
  clear bending magnet flag
}
```

Overflow bug



- Setup test checks F$mal
- Class3 == 0 → all good
- Class3 is 8 bits, inc on every setup test
- Every 256th time, rollover → skip collimiter check

Hit set button coincides with rollover:
* 25MeV turned on in field light (wrong) position

What were the “fixes”?

- Datent: another shared variable: “cursor not on command line”
- Overflow: Class3 set to fixed non-zero value instead of increment
- A handful of additional hardware interlocks
- (2+ years to get to this?)

Later extended to include:

- Better error messages
- Limited editing keys

How would you have fixed it?

What would you do differently?

Ostensible Causes

- Overconfidence in Software
- ***Confusing Reliability with Safety***
- No Defensive Design
- Failure to eliminate root causes
  - (piecemeal focus on individual errors)
- Complacency
- Unrealistic Risk Assessments
- Inadequate Investigation, Followup
- *Poor software engineering*
- Software Reuse
- Safe vs Friendly Interfaces
- Lack of Oversight / Standards

Poor software engineering

- Docs != afterthought
- Rigorous QA needed
- Avoid Hazardous coding idioms
- Audit trails designed in from beginning
- Need testing + formal analysis
- BetterUI/Manuals

The important memes for this class

- Reliability != Safety
  - How to precisely state your reliability / safety requirement?
  - How to state properties and requirements in general?
  - Risk assessment: super-important, super-sensitive
- Redundancy is critical: FT / defense in depth
- Many Ambient / Implicit Tradeoffs at work
  - Usability v. other properties (safety)
  - Programmability v. Performance
  - ***Should be surfaced, stated precisely, rendered quantifiable***
- General guidance
  - Don't mix an OS with your application
  - Use a high-level language: there are tradeoffs though, right?

Therac-25 is a system trying to satisfy a number of properties without ever formally stating what those properties are.

In subsequent readings:
What properties are pursued?
Are they implicit/explicit?
How are they achieved?