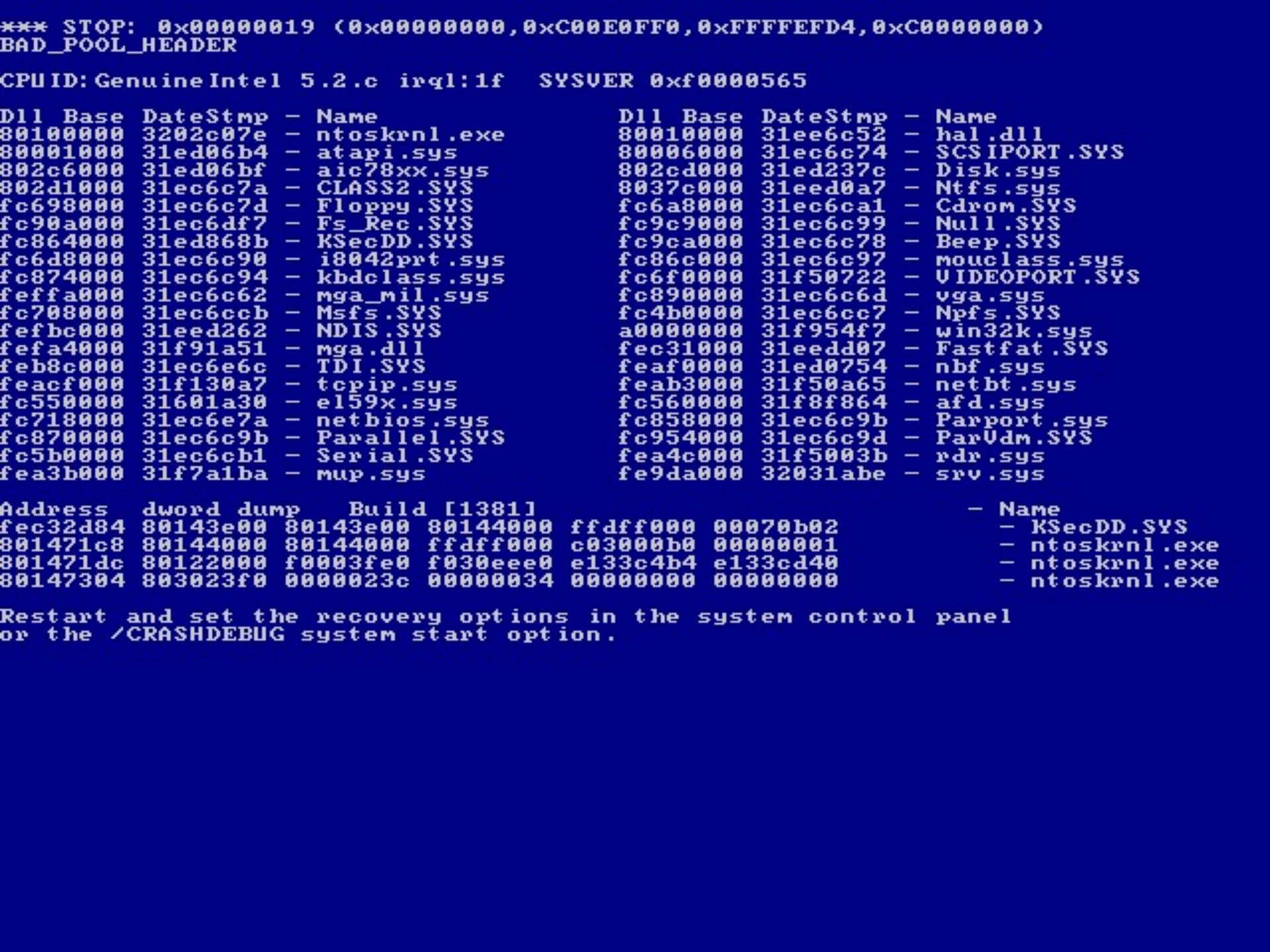
Steve's Research

Stephen Freund
Williams College

Announcements

- Lab today and tomorrow

- Project meetings Wed – Fri
  - Sign up for slots **today** if you have not
  - Meet in library lab


*** STOP: 0x00000019 (0x00000000,0xC00E0FF0,0xFFFFEFD4,0xC0000000)
BAD_POOL_HEADER

CPUID:GenuineIntel 5.2.c irql:1f SYSVER 0xf0000565

Dll Base DateStmp - Name
80100000 3202c07e - ntoskrnl.exe
80001000 31ed06b4 - atapi.sys
802c6000 31ed06bf - aic78xx.sys
802d1000 31ec6c7a - CLASS2.SYS
fc698000 31ec6c7d - Floppy.SYS
fc90a000 31ec6df7 - Fs_Rec.SYS
fc864000 31ed868b - KSecDD.SYS
fc6d8000 31ec6c90 - i8042prt.sys
fc874000 31ec6c94 - kbdclass.sys
feffa000 31ec6c62 - mga_mil.sys
fc708000 31ec6ccb - Msfs.SYS
fefbc000 31eed262 - NDIS.SYS
fefa4000 31f91a51 - mga.dll
feb8c000 31ec6e6c - TDI.SYS
feacf000 31f130a7 - tcpip.sys
fc550000 31601a30 - el59x.sys
fc718000 31ec6e7a - netbios.sys
fc870000 31ec6c9b - Parallel.SYS
fc5b0000 31ec6cb1 - Serial.SYS
fea3b000 31f7a1ba - mup.sys

Dll Base DateStmp - Name
80010000 31ee6c52 - hal.dll
80006000 31ec6c74 - SCSIPORT.SYS
802cd000 31ed237c - Disk.sys
8037c000 31eed0a7 - Ntfs.sys
fc6a8000 31ec6ca1 - Cdrom.SYS
fc9c9000 31ec6c99 - Null.SYS
fc9ca000 31ec6c78 - Beep.SYS
fc86c000 31ec6c97 - mouclass.sys
fc6f0000 31f50722 - VIDEOPORT.SYS
fc890000 31ec6c6d - vga.sys
fc4b0000 31ec6cc7 - Npfs.SYS
a0000000 31f954f7 - win32k.sys
fec31000 31eedd07 - Fastfat.SYS
feaf0000 31ed0754 - nbf.sys
feab3000 31f50a65 - netbt.sys
fc560000 31f8f864 - afd.sys
fc858000 31ec6c9b - Parport.sys
fc954000 31ec6c9d - ParVdm.SYS
fea4c000 31f5003b - rdr.sys
fe9da000 32031abe - srv.sys

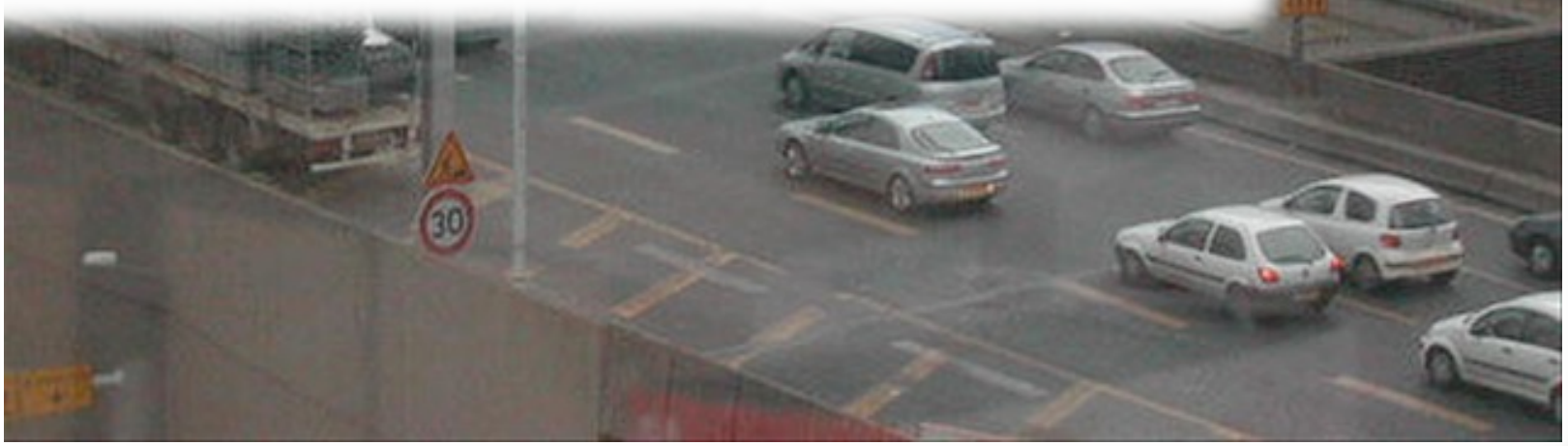
Address dword dump Build [1381] - Name
fec32d84 80143e00 80143e00 80144000 ffdff000 00070b02 - KSecDD.SYS
801471c8 80144000 80144000 ffdff000 c03000b0 00000001 - ntoskrnl.exe
801471dc 80122000 f0003fe0 f030eee0 e133c4b4 e133cd40 - ntoskrnl.exe
80147304 803023f0 0000023c 00000034 00000000 00000000 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option.

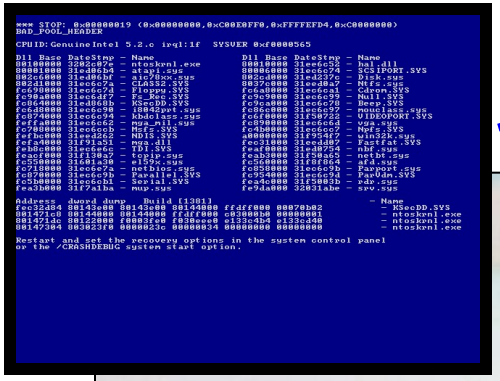
The Blue Screen of Death


the Bay
the Bay


America
sit
8PM


30

USS Yorktown



- Smart Ship
  - 27 PCs
  - Windows NT 4.0

- September 21, 1997:
  - data entry error caused a "Divide-By-0" error
  - entire system failed
  - ship dead in the water for over 2 hours

[Wired 1997]


Ariane 5 Rocket
June 4, 1996
$800 million software failure

Mars Climate Orbiter

Purpose: Collect data. Relay signals from Mars Polar Lander ($165M)

Failure: Smashed into Mars (1999)

Bug: Failed to convert English to metric units

Mars Polar Lander

Purpose: Lander to study the Mars climate ($120M)

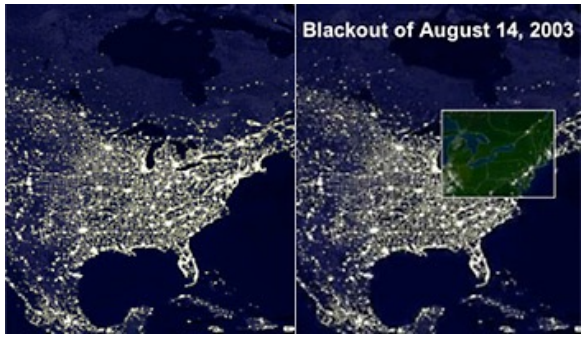
Failure: Smashed into Mars (2000)

Bug: Spurious signals from sensors caused premature engine shutoff

North East Power Failure

Failure: Power grid failed across much of the North East. $6B losses (2001)

Bug: Timing bug in Ohio power plant

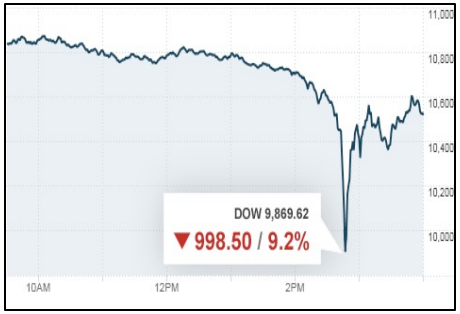


Online Trading Software

Purpose: automatic high-frequency trading

Failure: DOW drops 9.2%, equity markets collapse (2010)

Bug: Bad modeling, and no fail-stops to prevent flooding market with sell orders

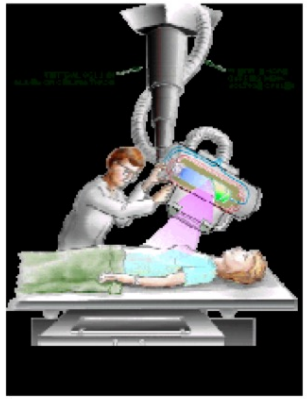


Therac25 Radiation Therapy

Purpose: Computer-controlled radiation therapy machine

Failure: gave fatal radiation doses to 2 cancer patients (1986)

Bug: race condition (timing bug)



Patriot Missile

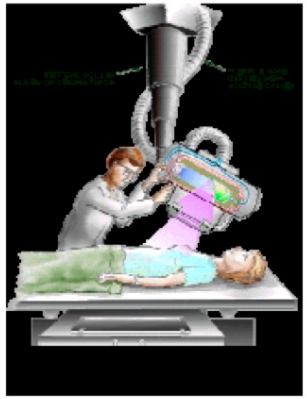
Purpose: Intercept incoming missiles

Failure: missed SCUD missile that killed 28 US soldiers (1991)

Bug: incorrect calculation of distance to target

Therac25 Radiation Therapy

Purpose: Computer-controlled radiation therapy machine



Failure: gave fatal radiation doses to 2 cancer patients (1986)

Bug: race condition (timing bug)

Patriot Missile

Purpose: Intercept incoming missiles

Failure: missed SCUD missile that killed 28 US soldiers (1991)

Bug: incorrect calculation of distance to target

Tesla

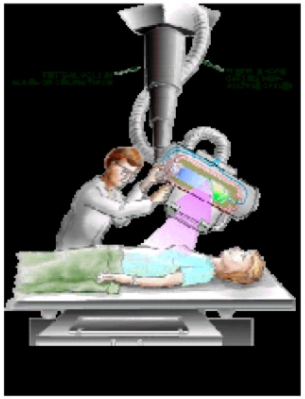
Purpose: Self-Driving Cars

Failure: Fatal Crash (2016)

Bug: Failed to distinguish a white tractor-trailer crossing the highway against a bright sky. (Other fatal accidents have followed…)

Therac25 Radiation Therapy

Purpose: Computer-controlled radiation therapy machine



Failure: gave fatal radiation doses to 2 cancer patients (1986)

Bug: race condition (timing bug)

Tesla

Purpose: Self-Driving Cars

Failure: Fatal Crash (2016)

Bug: Failed to distinguish a white tractor-trailer crossing the highway against a bright sky. (Other fatal accidents have followed…)

Feb 1, 2022: Tesla recalls 54,000 vehicles due to **software letting them roll through stop signs** without coming to a complete halt.

Feb 3, 2022: Tesla recalls 800,000 vehicles due software bug related to **seat belt reminders**.

April 29, 2022: Tesla recalls 63,000 cars due to a software bug making it **hard to tell how fast the car is going**.

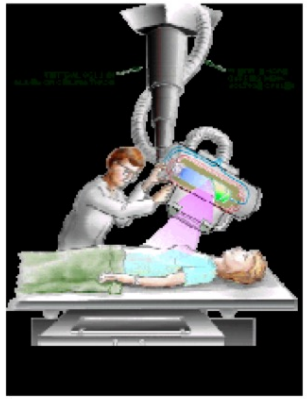
May 11, 2022: Tesla recalls130,000 cars due to software bug leading to **overheating in display system.**

Nov 3, 2022: Tesla recalls 11,000 cars due to bug causing vehicles to **activate forward-collision warnings and activate the emergency brakes for no reason.**

Nov 8, 2022: Tesla recalls 40,000 cars due to software update leading to **power steering failure**.

Therac25 Radiation Therapy

Purpose: Computer-controlled radiation therapy machine



Failure: gave fatal radiation doses to 2 cancer patients (1986)

Bug: race condition (timing bug)

Patriot Missile

Purpose: Intercept incoming missiles

Failure: missed SCUD missile that killed 28 US soldiers (1991)

Bug: incorrect calculation of distance to target

Tesla

Purpose: Self-Driving Cars

Failure: Fatal Crash (2016)

Bug: Failed to distinguish a white tractor-trailer crossing the highway against a bright sky. (Other fatal accidents have followed…)

Heartbleed SSL Attack

Purpose: OpenSSL is widely-used cryptographic library.

Failure: Library could leak secret information, including keys. (2014)

Bug: Buffer overrun

Buffer Overruns

[2]:
```
def f(array,index):
    array[index] = 42
```

[3]:
```
elems = make_array(1,2,3,4,5,6,7,8,9,10)
x = 100

print(x)
f(elems,6)
print(x)
```

```
100
100
```

Buffer Overruns

[2]:
```
def f(array,index):
    array[index] = 42
```

[3]:
```
elems = make_array(1,2,3,4,5,6,7,8,9,10)
x = 100
```

[4]:
```
print(x)
f(elems,11)
print(x)
```

```
100
```

```
---------------------------------------------------------------------------
IndexError: index 11 is out of bounds for axis 0 with size 10
```

Buffer Overruns

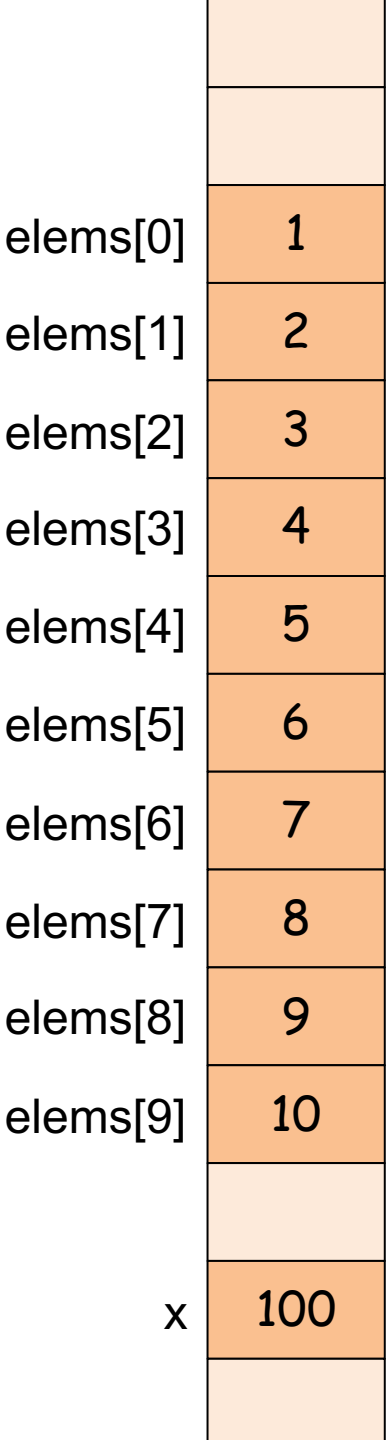
C Code!

```
void f(int array[], int index) {
  array[index] = 42;
}

int main() {
  int x = 100;
  int elems[10] = { 1,2,3,4,5,6,7,8,9,10 };

  printf("%d\n", x);
  f(elems, 6);
  printf("%d\n", x);
  f(elems, 11);
  printf("%d\n", x);
}
```

```
$ gcc array.c
$ ./a.out
100
100
42
```

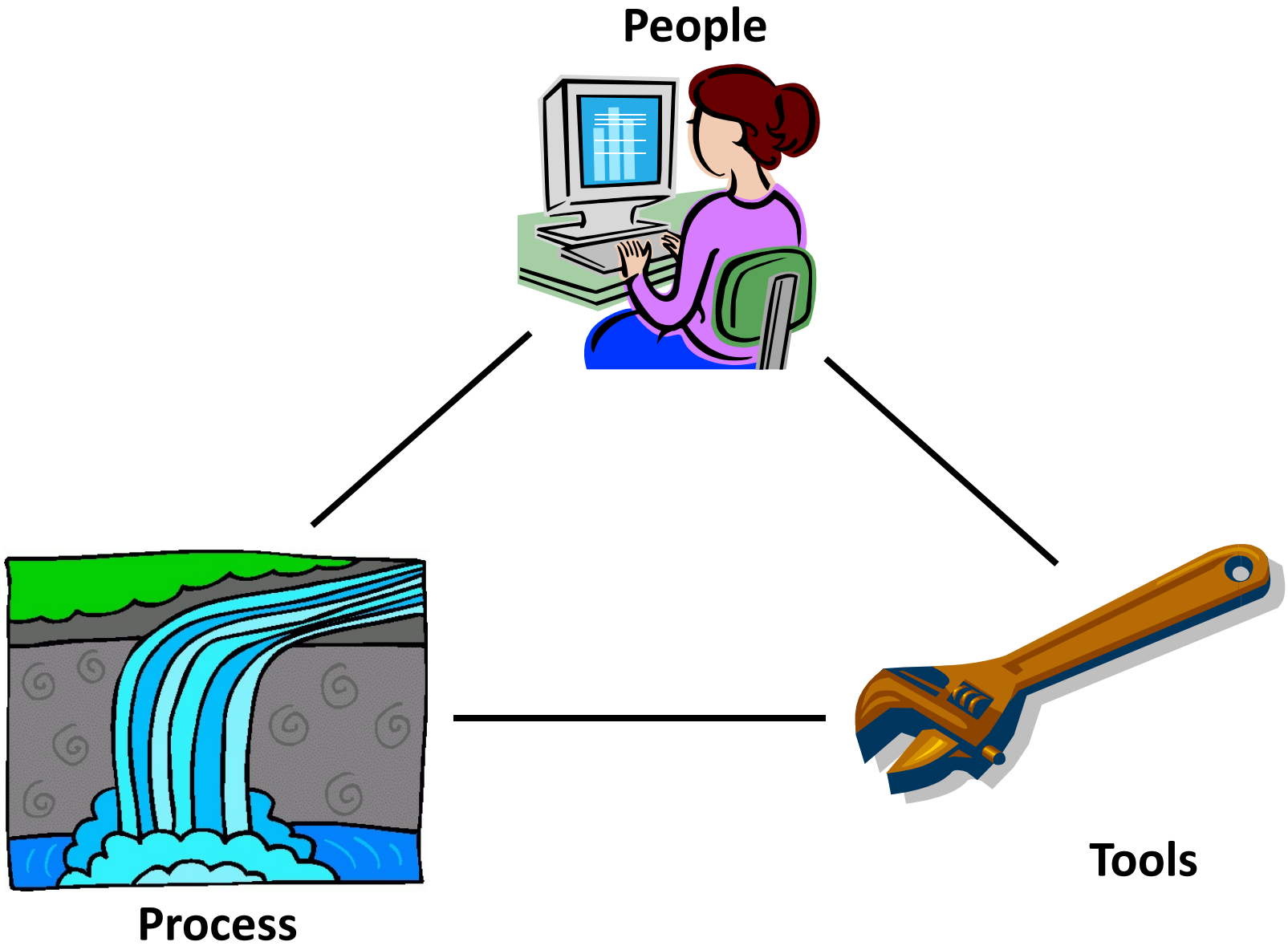
Buffer Overruns

```
void f(int array[], int index) {
  array[index] = 42;
}

int main() {
  int x = 100;
  int elems[10] = { 1,2,3,4,5,6,7,8,9,10 };

  printf("%d\n", x);
  f(elems, 6);
  printf("%d\n", x);
  f(elems, 11);
  printf("%d\n", x);
}
```



https://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Managing Software Complexity



Research on Program Checkers

Identify Type of Bug

- Bad unit conversion
- Buffer overrun
- Data Race
- …

Design Checking Tool

- *static or dynamic?*
- precision?
- scalability?
- performance?
- usability?

Validate Technique

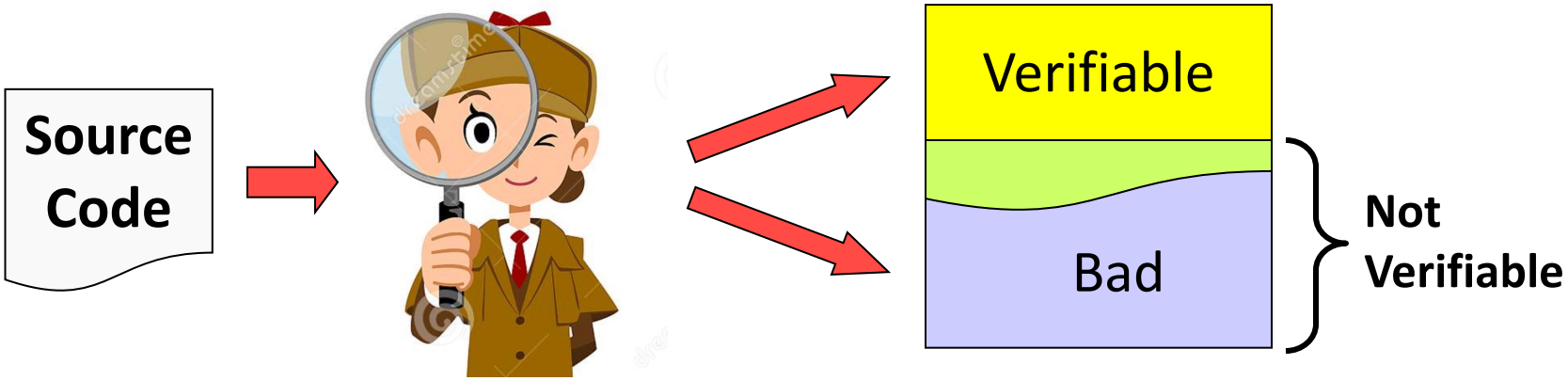
- check real software
- find bugs...

Source Code (Static) Checkers

Good Program	Has No Buffer Overruns
Bad Program	Has Buffer Overrun

- **No algorithm can precisely compute if a program is "Good" or "Bad"**
  - Undecidability of the Halting Problem [Turing 1936]

Source Code (Static) Checkers



Good Program	Has No Buffer Overruns
Bad Program	Has Buffer Overrun
Verifiable Program	Can **Prove** No Buffer Overruns

\+ Catch many errors prior to testing

− Must reject some good programs...

Dynamic Checkers

+ Can discern Good vs. Bad precisely, but…

– only during the tests performed

– Performance

Identify Type of Bug

- Bad unit conversion
- Buffer overrun
- Data Race
- …

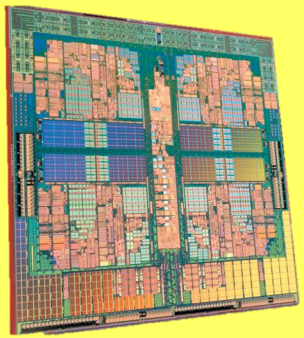
Design Checking Tool

- static or dynamic?
- precision?
- scalability?
- performance?
- usability?

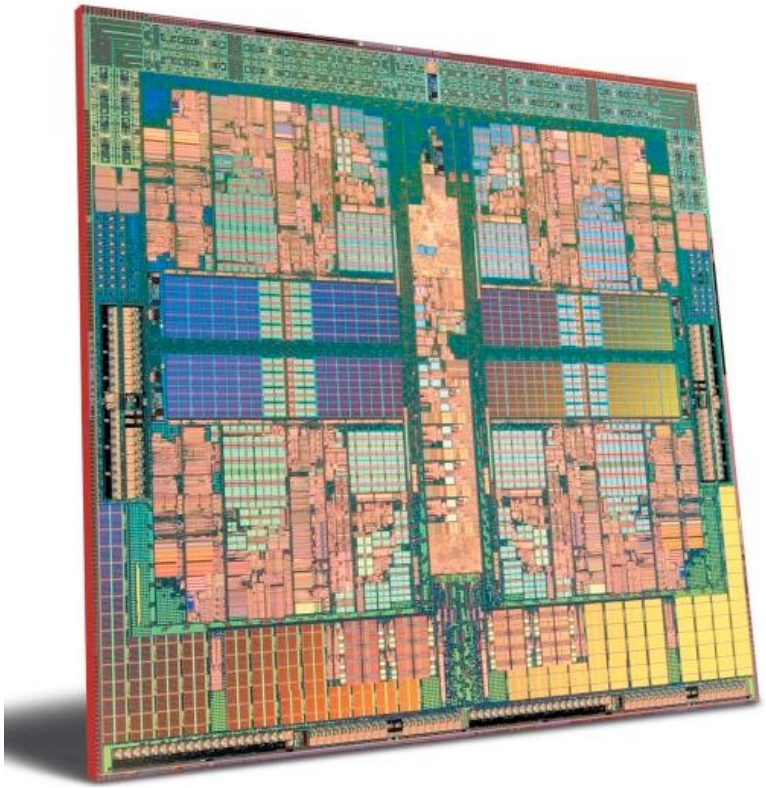
Validate Technique

- check real software
- find bugs…

New Languages, Programming Models



Multithreading and Multicore CPUs



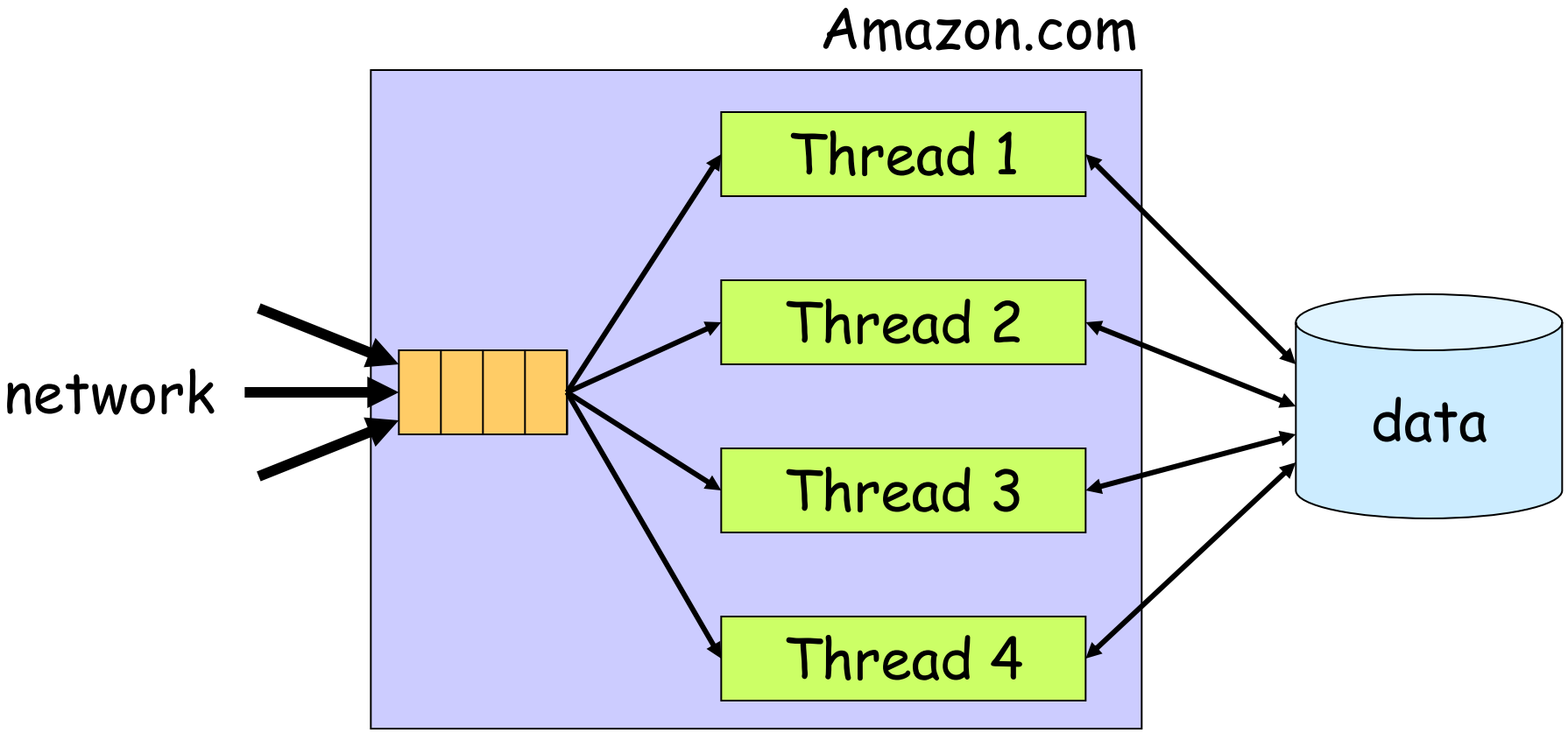
Concurrent Programming With Threads

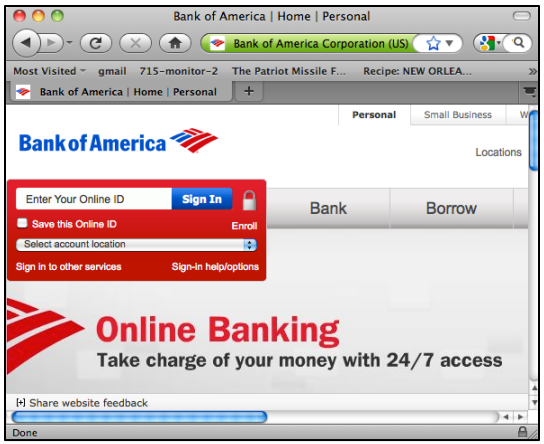
Concurrent Programming With Threads

```
y_hat = a * table.column('x') + b
```

Divide array into four pieces and do multiplications and additions for each piece on a different thread

Concurrent Programming With Threads




Bank of America | Home | Personal
Bank of America Corporation (US)
Most Visited
gmail
715-monitor-2
The Patriot Missile F...
Recipe: NEW ORLEA...
Bank of America | Home | Personal
Personal
Small Business
Bank of America
Locations
Enter Your Online ID
Sign In
Save this Online ID
Enroll
Select account location
Sign in to other services
Sign-in help/options
Bank
Borrow
Online Banking
Take charge of your money with 24/7 access
Share website feedback
Done

Bank Server


Withdrawal

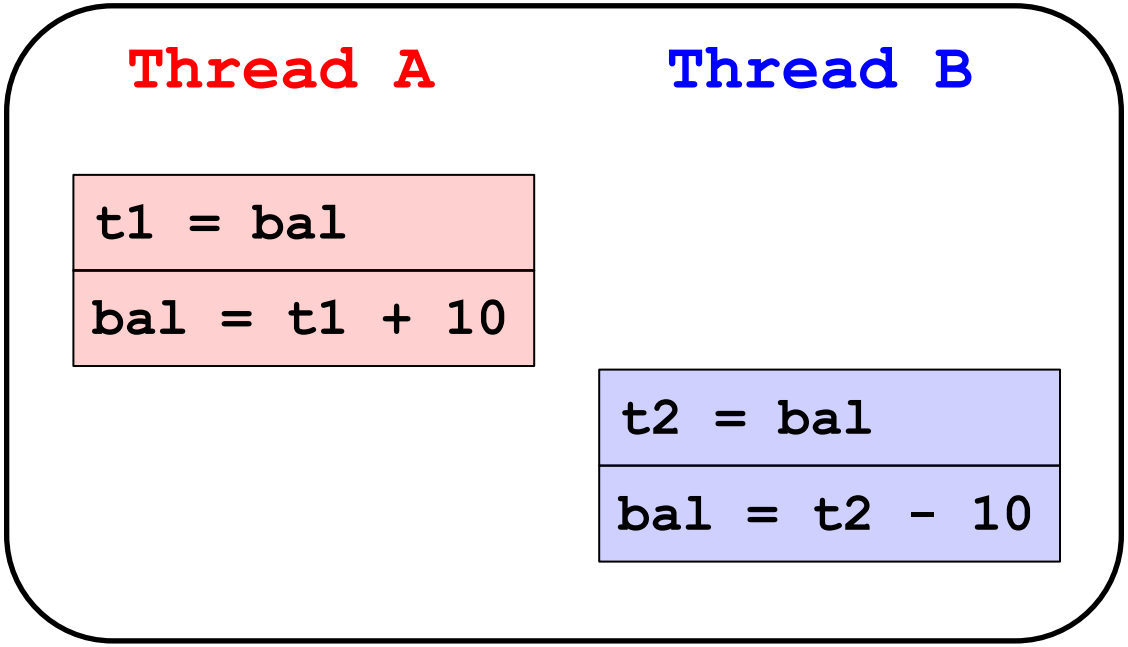
Thread Interference

- Race Conditions

  two concurrent unsynchronized accesses, at least one write

```
Thread A
  ...
  t1 = bal;
  bal = t1 + 10;
  ...
```

```
Thread B
  ...
  t2 = bal;
  bal = t2 - 10;
  ...
```



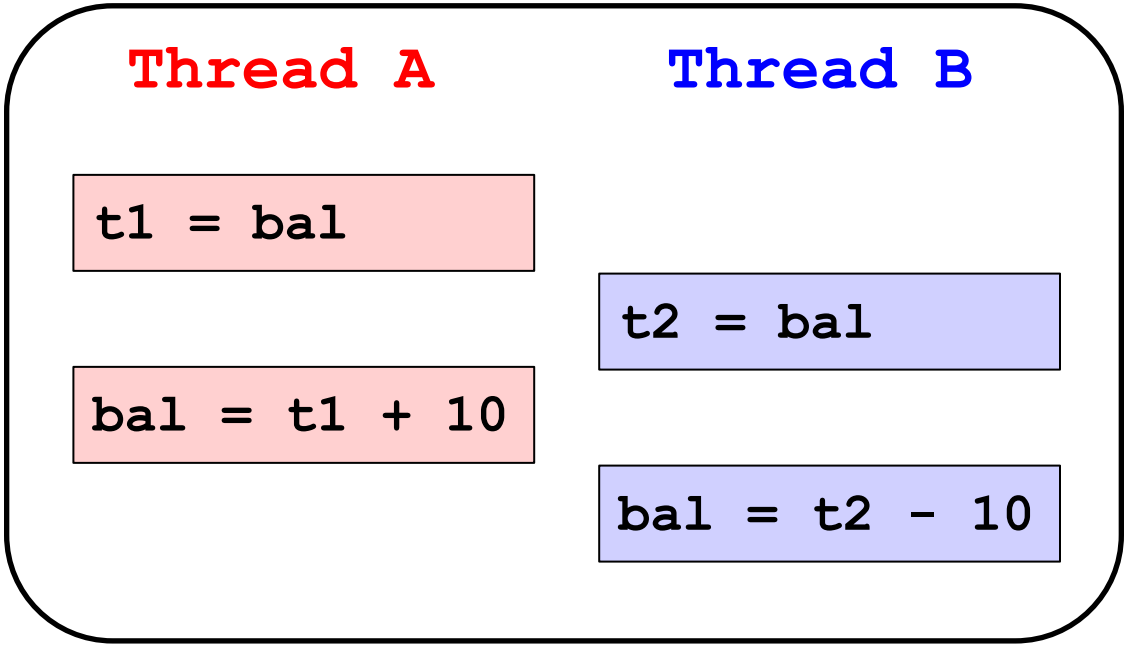
Thread Interference

- Race Conditions

  two concurrent unsynchronized accesses, at least one write

```
Thread A
  ...
  t1 = bal;
  bal = t1 + 10;
  ...
```

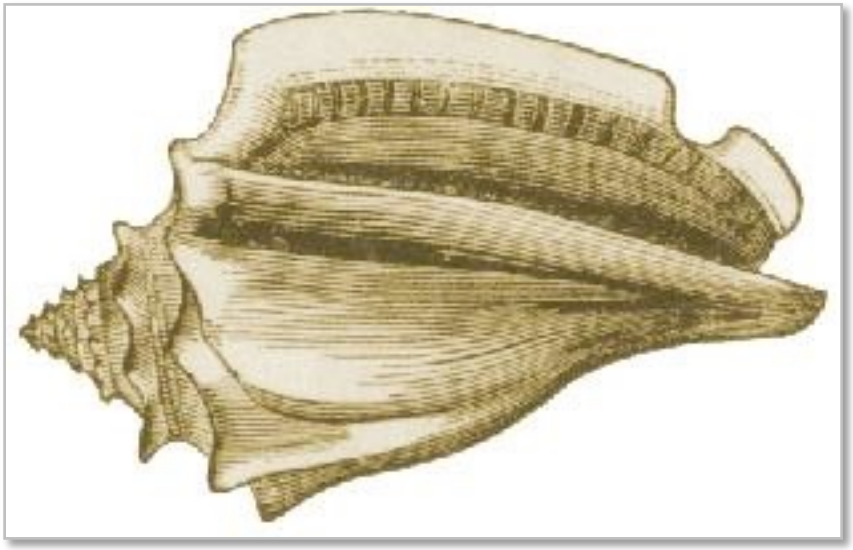
```
Thread B
  ...
  t2 = bal;
  bal = t2 - 10;
  ...
```



Controlling Thread Interference: Mutual Exclusion Locks

```
Thread A
  acq(m);
   t1 = bal;
   bal = t1 + 10;
  rel(m);
```

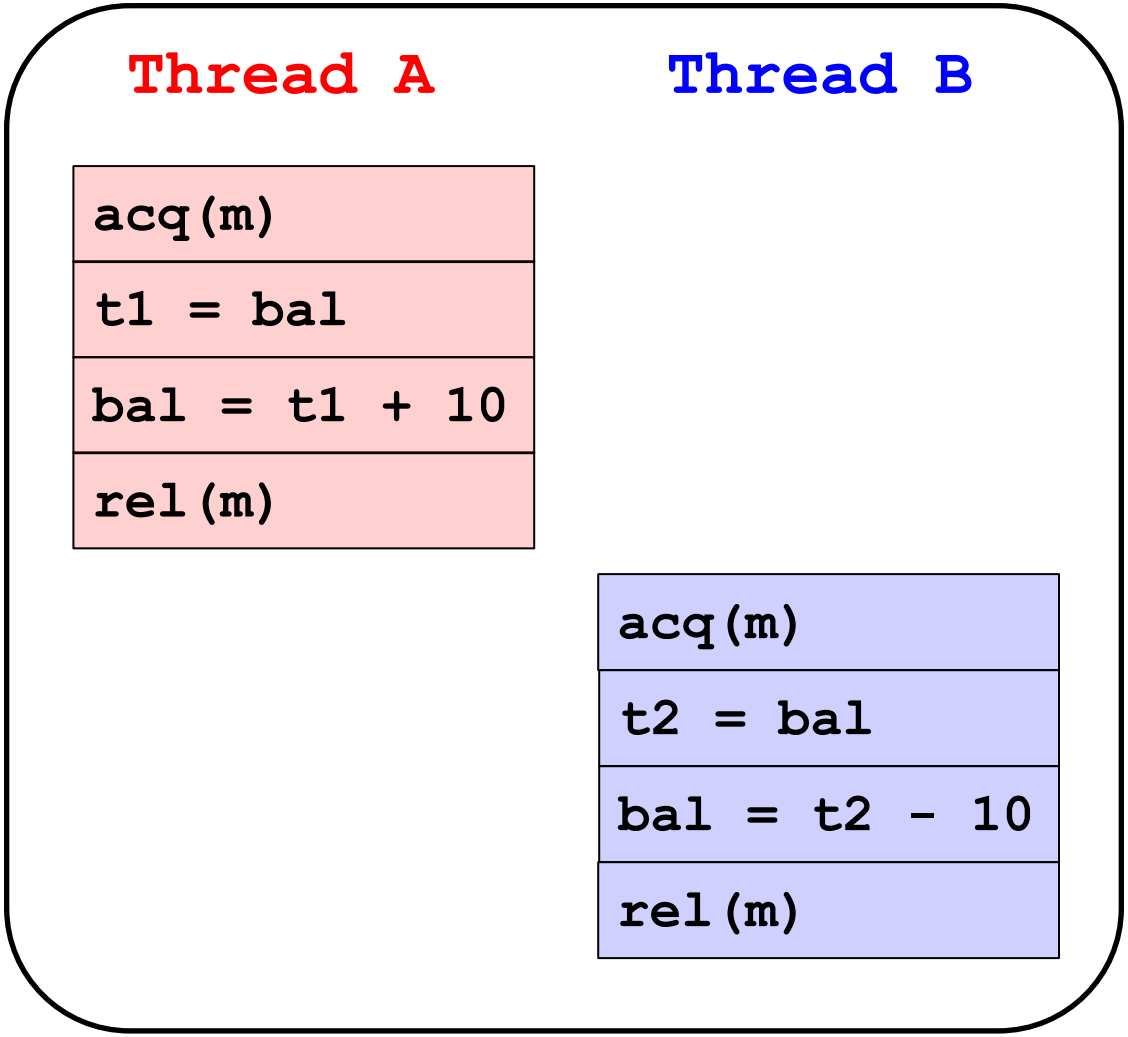
```
Thread B
  acq(m);
   t2 = bal;
   bal = t2 - 10;
  rel(m);
```



Controlling Thread Interference: Mutual Exclusion Locks

```
Thread A
  acq(m);
   t1 = bal;
   bal = t1 + 10;
  rel(m);
```

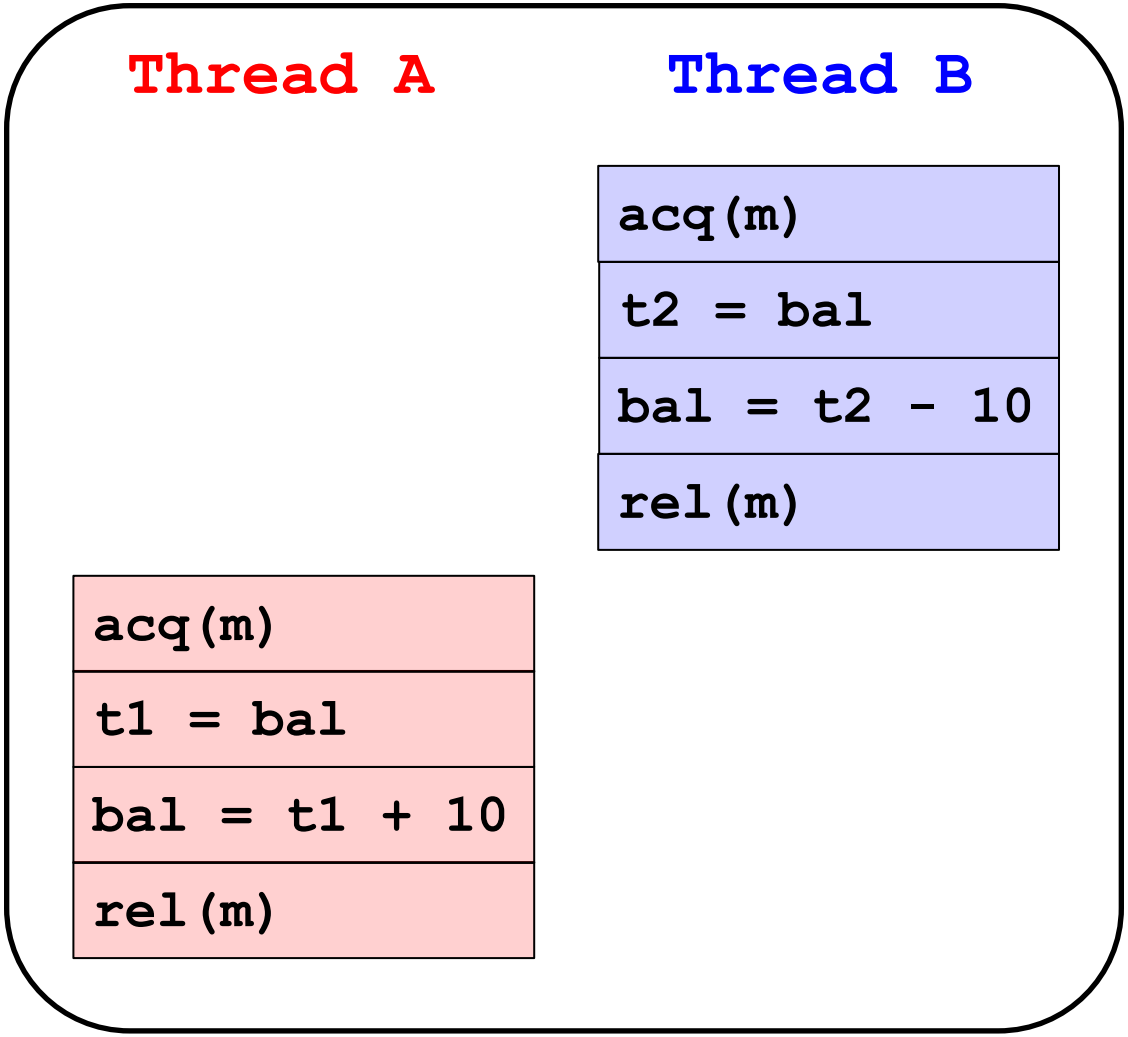
```
Thread B
  acq(m);
   t2 = bal;
   bal = t2 - 10;
  rel(m);
```



Controlling Thread Interference: Mutual Exclusion Locks

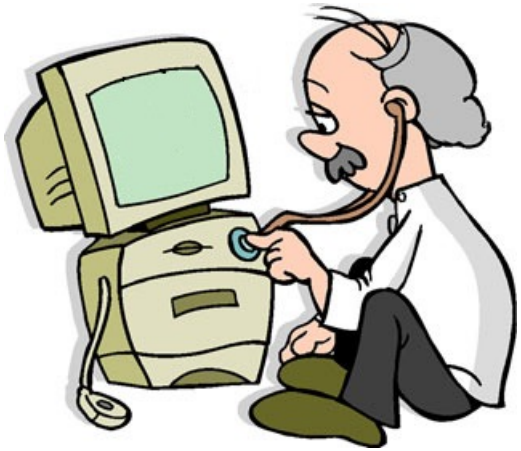
```
Thread A
  acq(m);
   t1 = bal;
   bal = t1 + 10;
  rel(m);
```

```
Thread B
  acq(m);
   t2 = bal;
   bal = t2 - 10;
  rel(m);
```



FastTrack '10, RedCard '13, SlimState '15, BigFoot '17

Executable Target



Data Race on var x

RccJava '02

```
class A {
 guarded_by m1
 int x;

 requires m1, m2
 void f() {
  synchronized …
  …
}
```

Data Race on var x

35

An Introduction to Programming with Threads

by Andrew D. Birrell

January 6, 1989

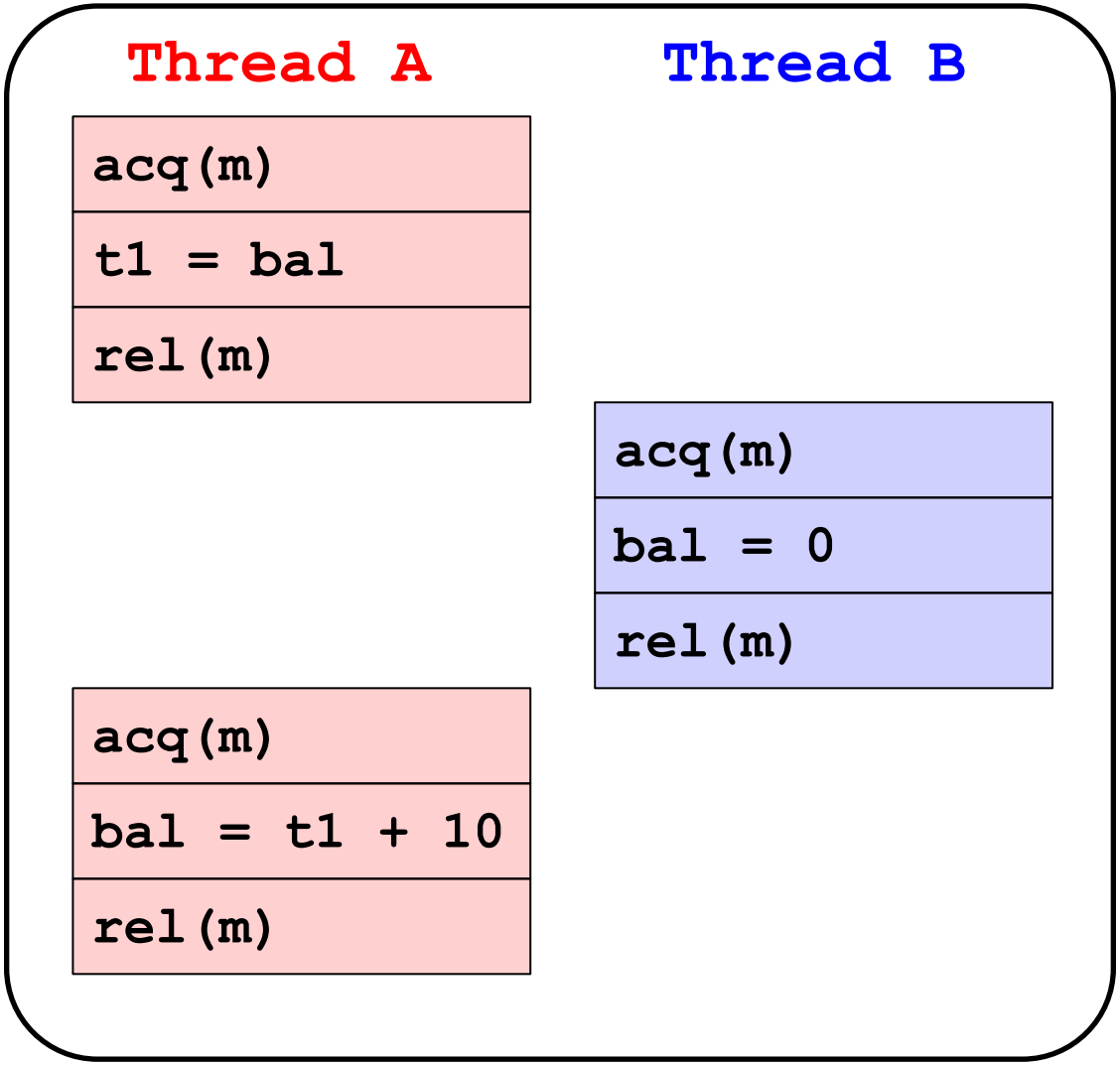
digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Race Freedom is not Enough...

```
Thread A
  ...
  acq(m);
  t1 = bal;
  rel(m);

  acq(m);
  bal = t1 + 10;
  rel(m);
```

```
Thread B
  ...
  acq(m);
  bal = 0
  rel(m);
```



Controlling Thread Interference: Enforce Atomicity

Atomic method must behave as if it executed serially, without interleaved operations of other thread

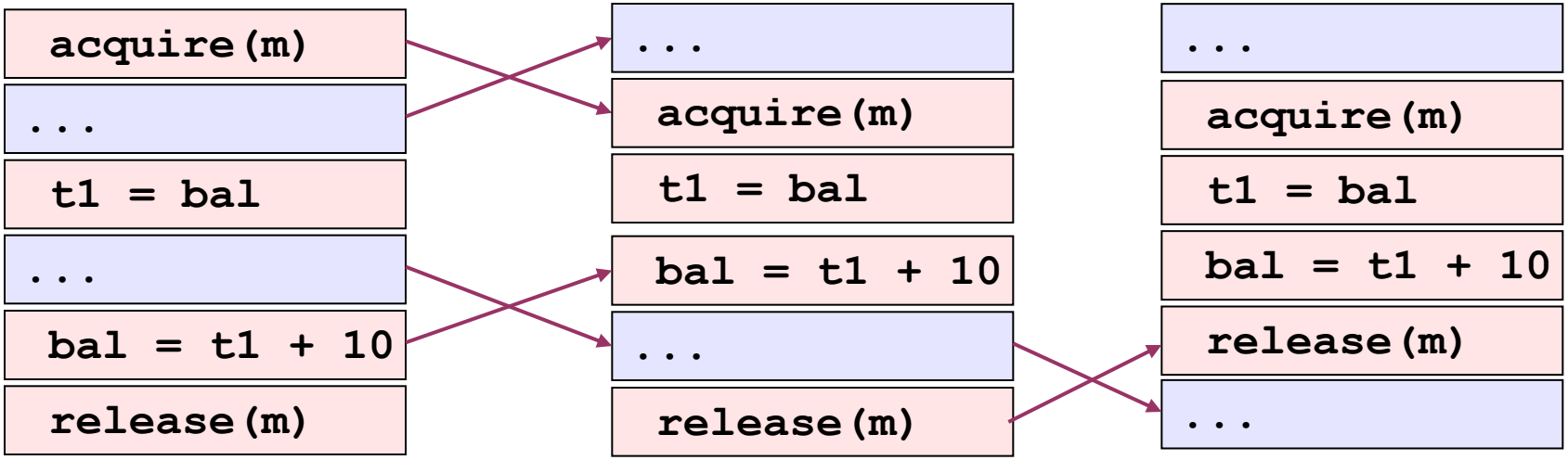
```
atomic void copy() {
   x = 0;
    thread interference?
   while (x < len) {
     thread interference?
     tmp = a[x];
     thread interference?
     b[x] = tmp;
     thread interference?
     x++;
     thread interference?
   }
}
```

Controlling Thread Interference: Enforce Atomicity

Atomic method must behave as if it executed serially, without interleaved operations of other thread

```
atomic void copy() {
  x = 0;
  thread interference?
  while (x < len) {
    thread interference?
    tmp = a[x];
    thread interference?
    b[x] = tmp;
    thread interference?
    x++;
    thread interference?
  }
}
```

Theory of Reduction [Lipton 76]



R	Right-mover	Acquire
L	Left-mover	Release
B	Both-mover	Race-Free Access
N	Non-mover	Racy Access

Serializable blocks have the pattern: R* [N] L*

Examples

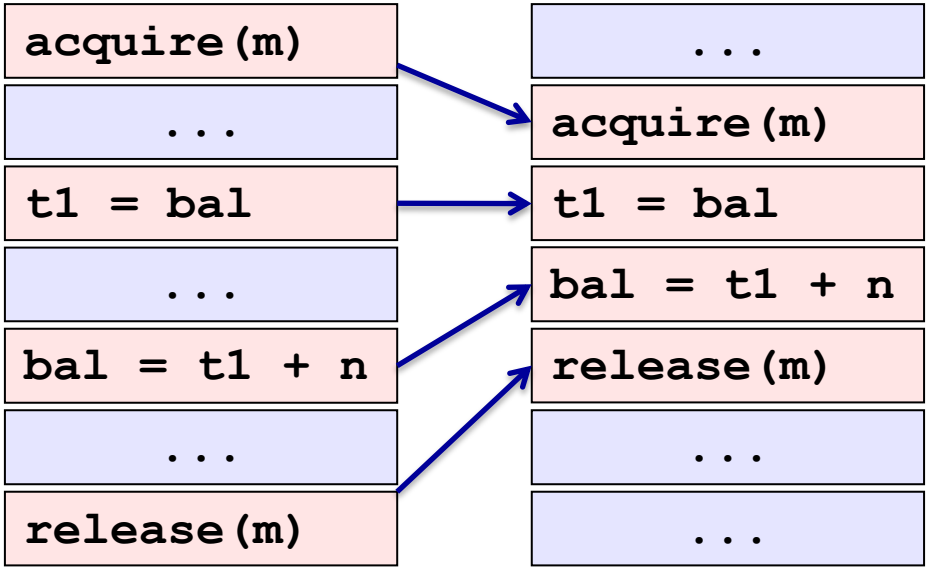
```
void deposit(int n) {
  synchronized(m) {
    t1 = bal;
    bal = t1 + n;
  }
}
```

R

B

B

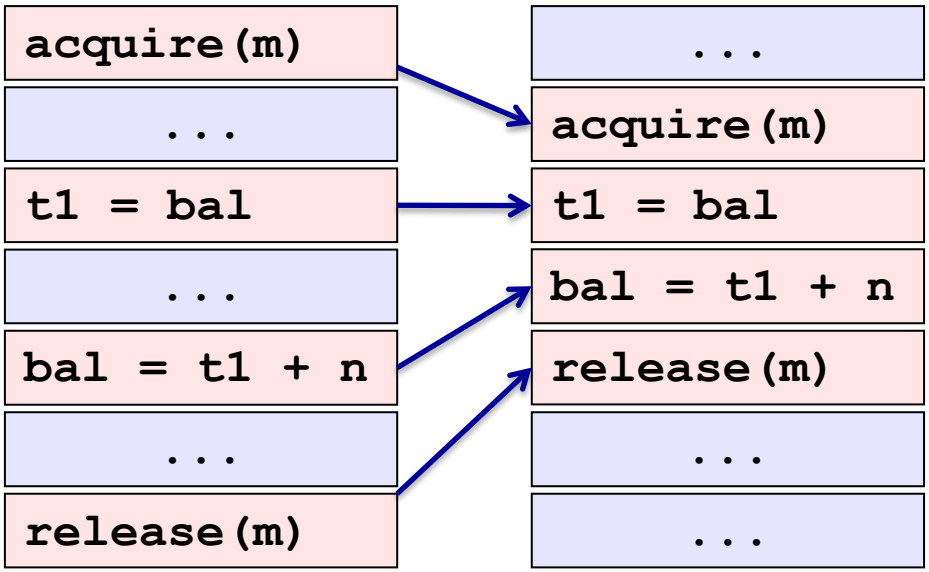
L



(R* [N] L*)

Examples

```
void deposit(int n) {
  synchronized(m) {
    t1 = bal;
    bal = t1 + n;
  }
}
```




```
void deposit(int n) {
  synchronized(m) {
    t1 = bal;
  }
  synchronized(m) {
    bal = t1 + n;
  }
}
```

R B L R B L

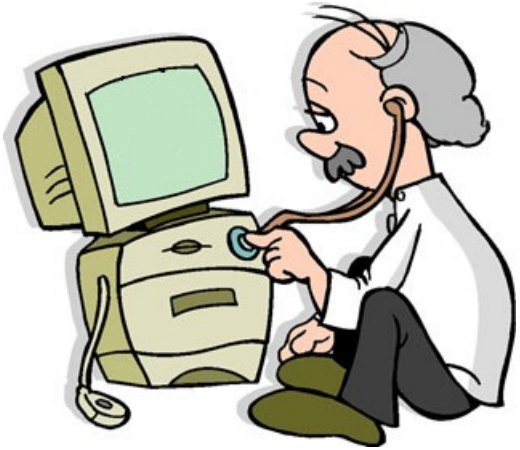
```
acquire(m)
t1 = bal
release(m)
...
acquire(m)
bal = t1 + n
release(m)
```

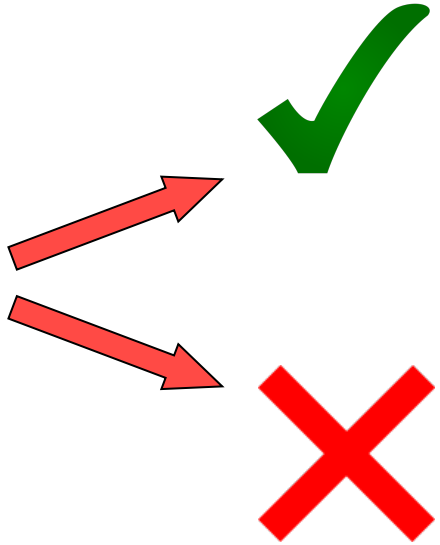
R M L R M L

(R* [N] L*)

Atomizer '04

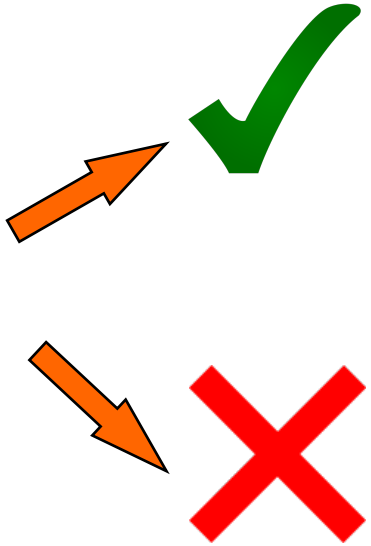
Executable Target (w/ atomic specs)





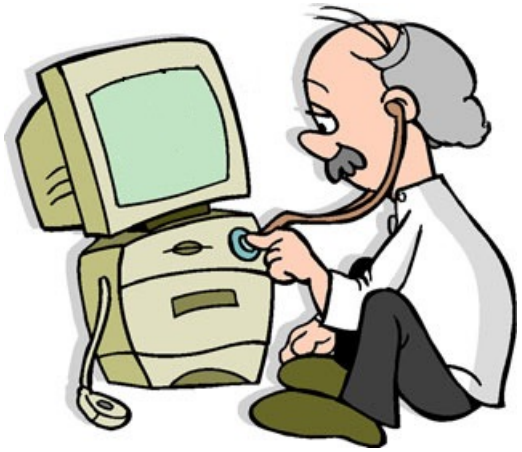
Bohr '08

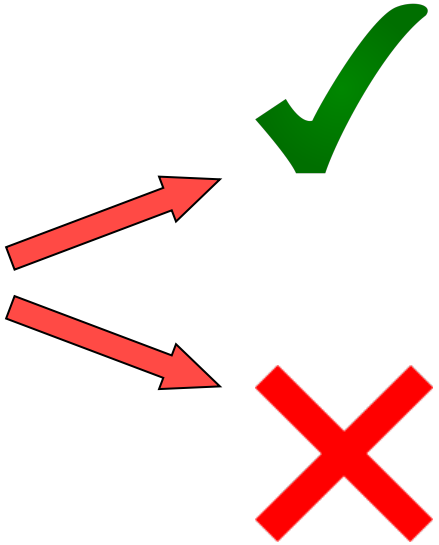
```
class A {
 race-free int x;

 atomic void m(){
  synchronized …
  …
 }
}
```



Atomizer '04

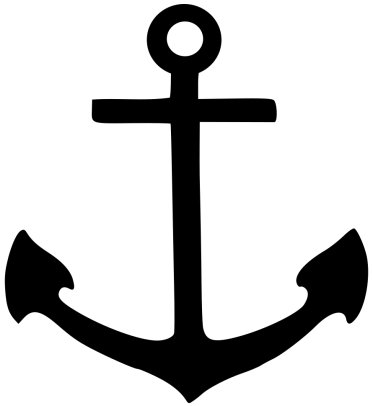
Executable Target (w/ atomic specs)

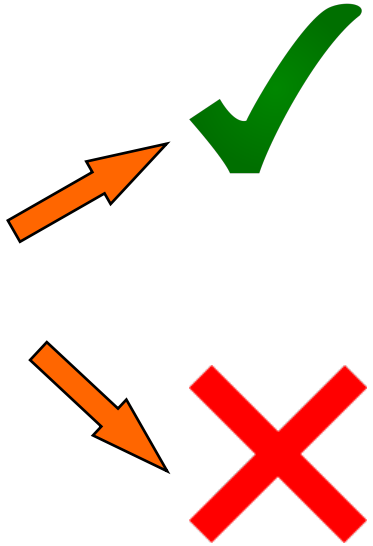




Anchor '20

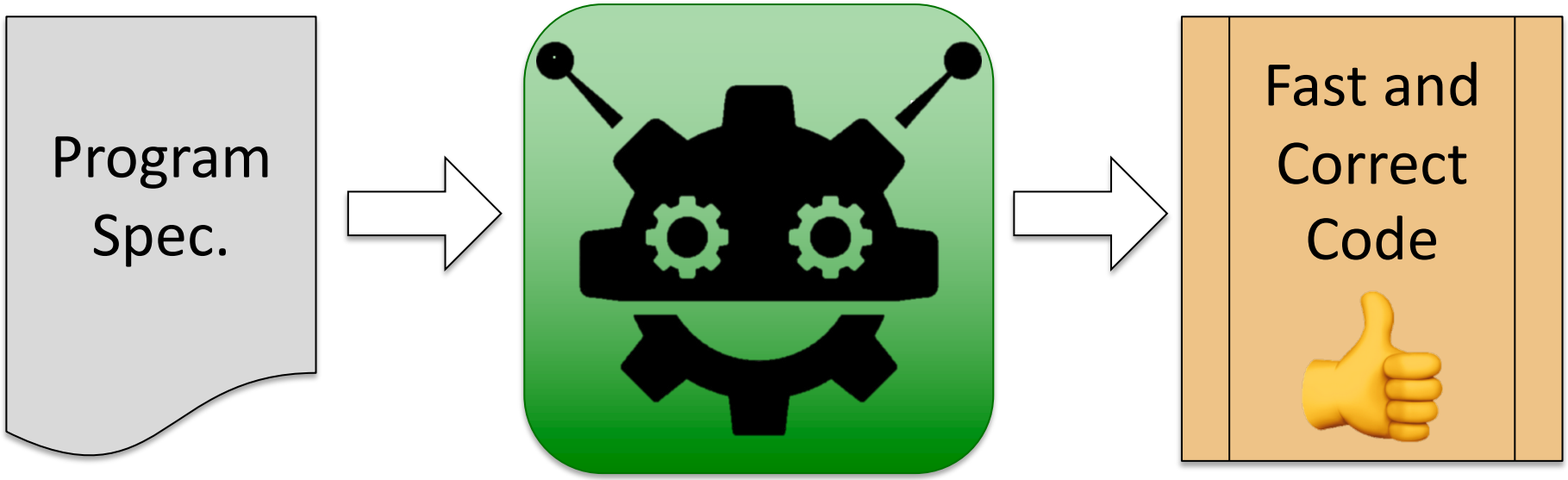
```
class A {
 race-free int x;

 atomic void m(){
  synchronized …
  …
 }
}
```



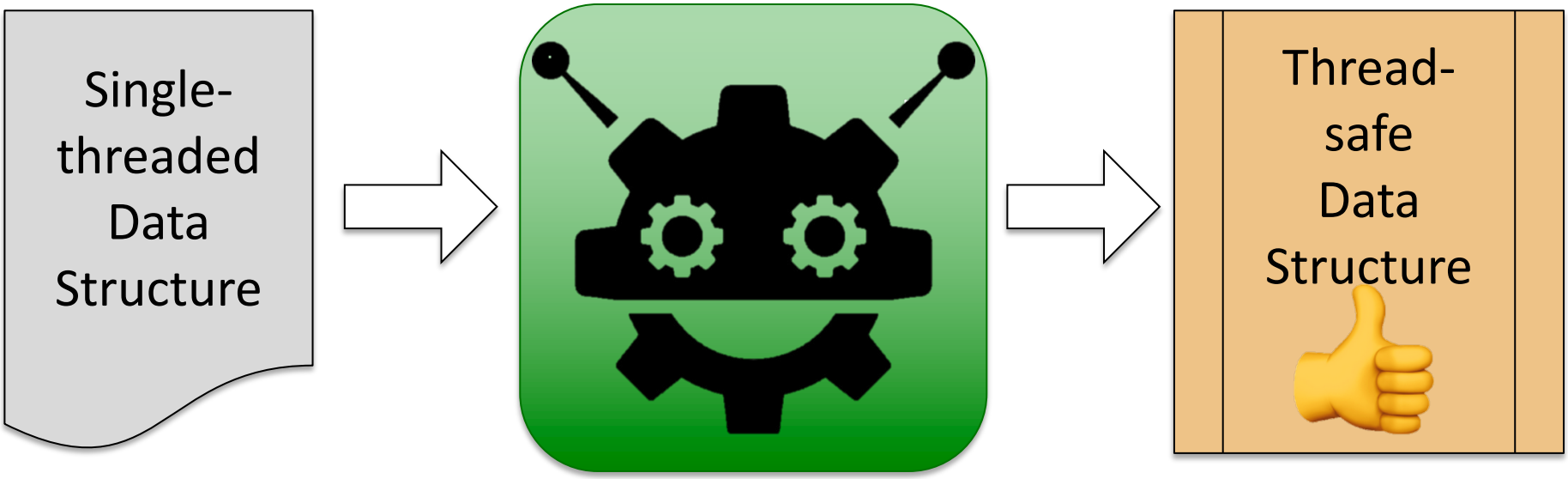


Traditional Software Process

Program Synthesis



Program Synthesis



- How to generate candidate versions?
- How to verify candidates are correct?
- How to pick most performant?

Programming Languages And Analysis Tools

- language design
- theoretical foundations
- proving theorems
- systems development
- performance modeling
- experimental validation