Data Provenance Tracking for Concurrent Programs

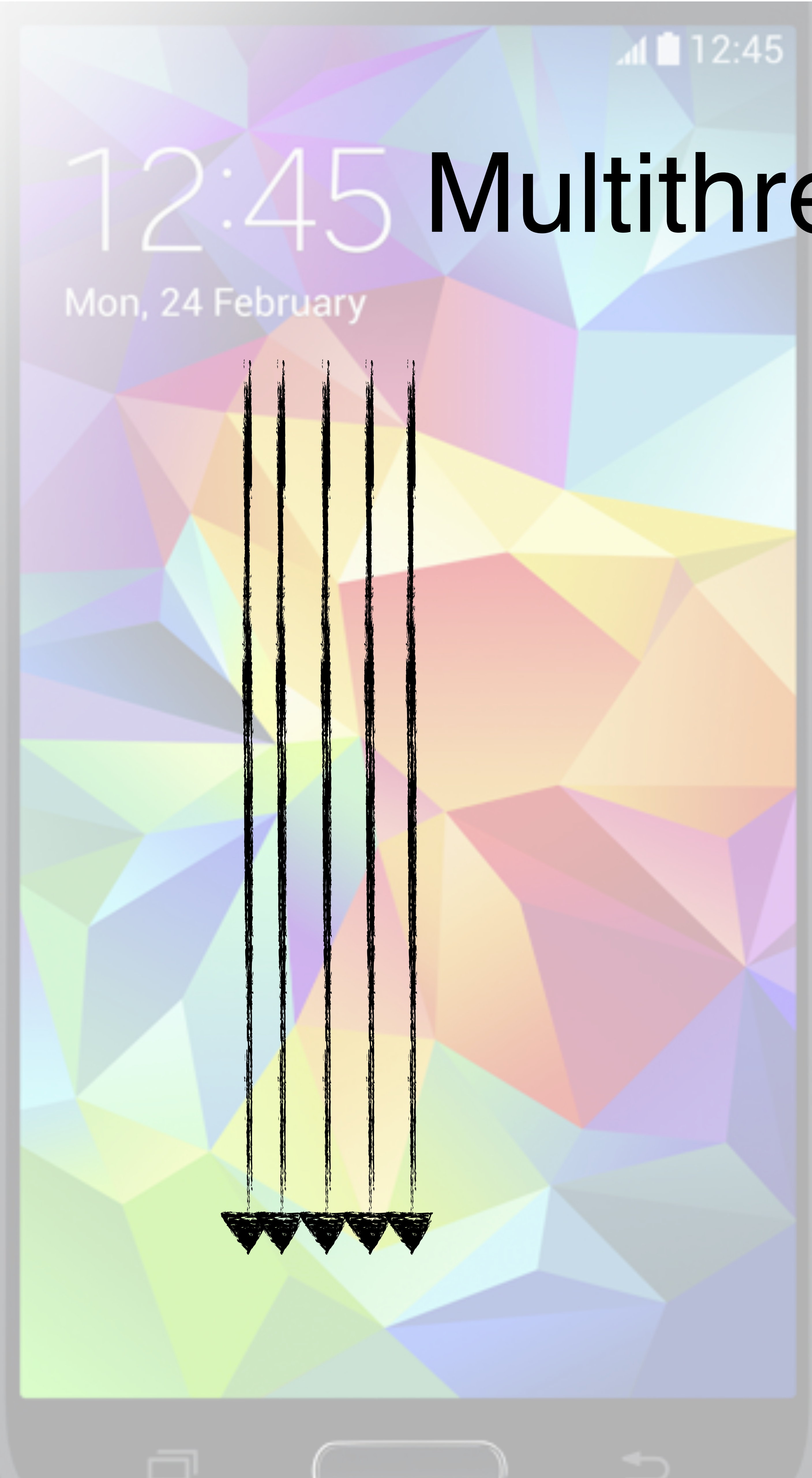
Brandon Lucia | Carnegie Mellon University, Dept. of ECE

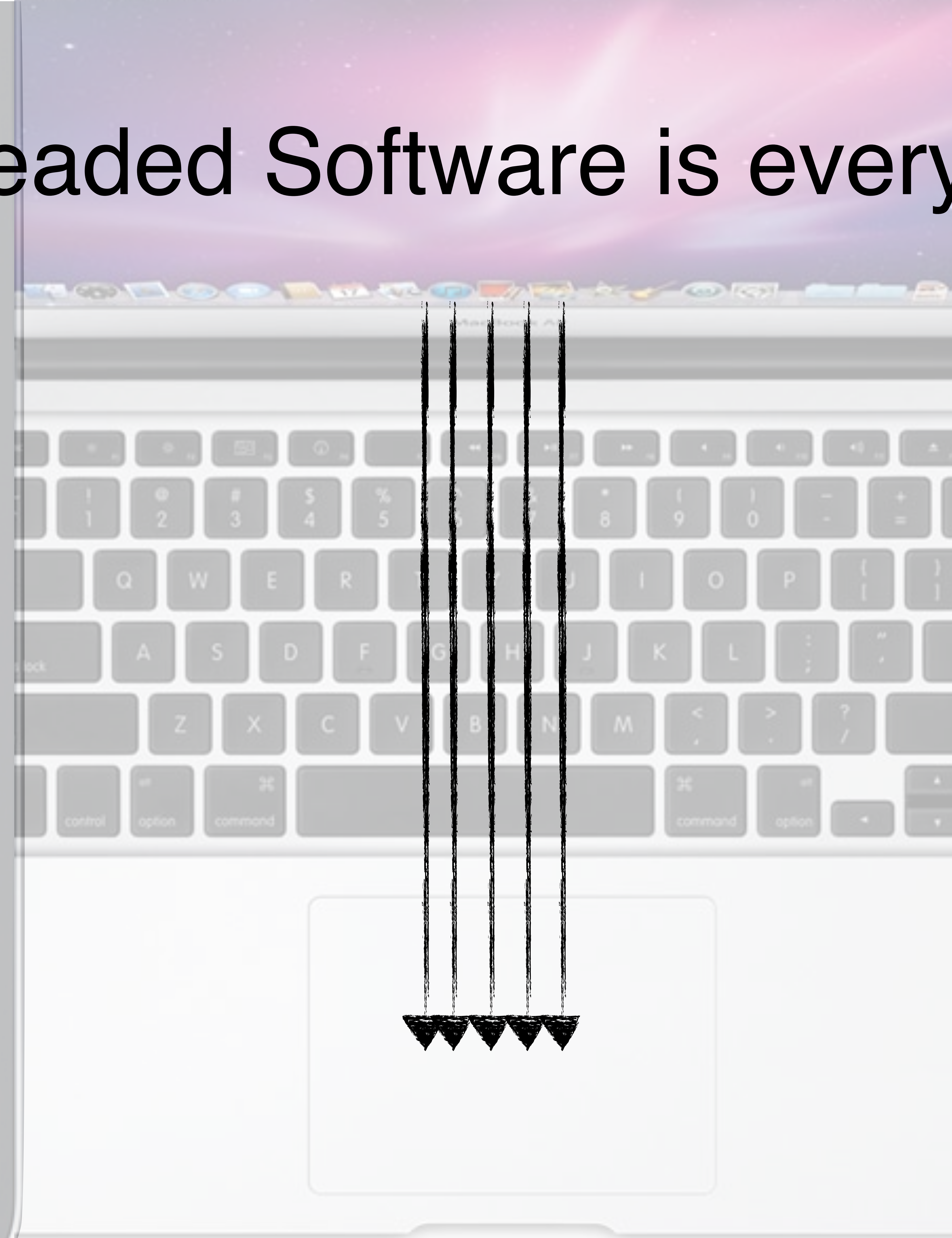
*work done in cooperation with **Luis Ceze @ University of Washington, Dept. of CSE***

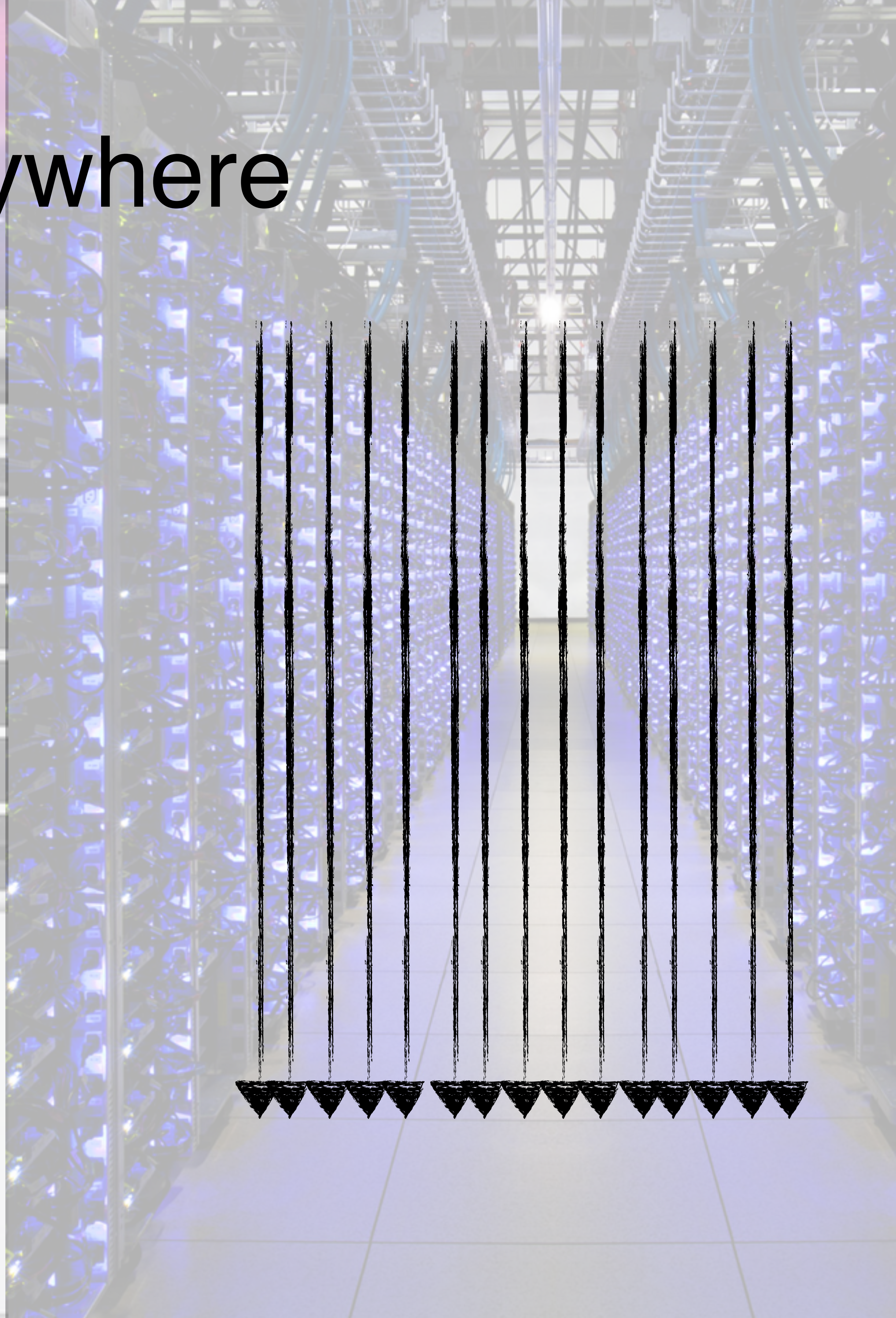
12:45
12:45
Mon, 24 February

MacBook Air
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
control
option
command
command
option

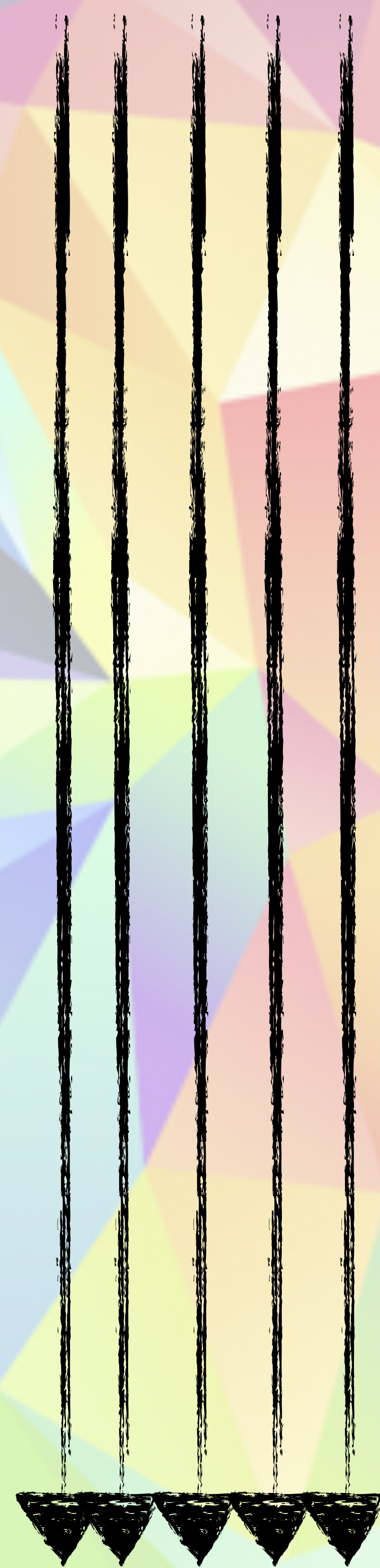
Multithreaded Software is everywhere

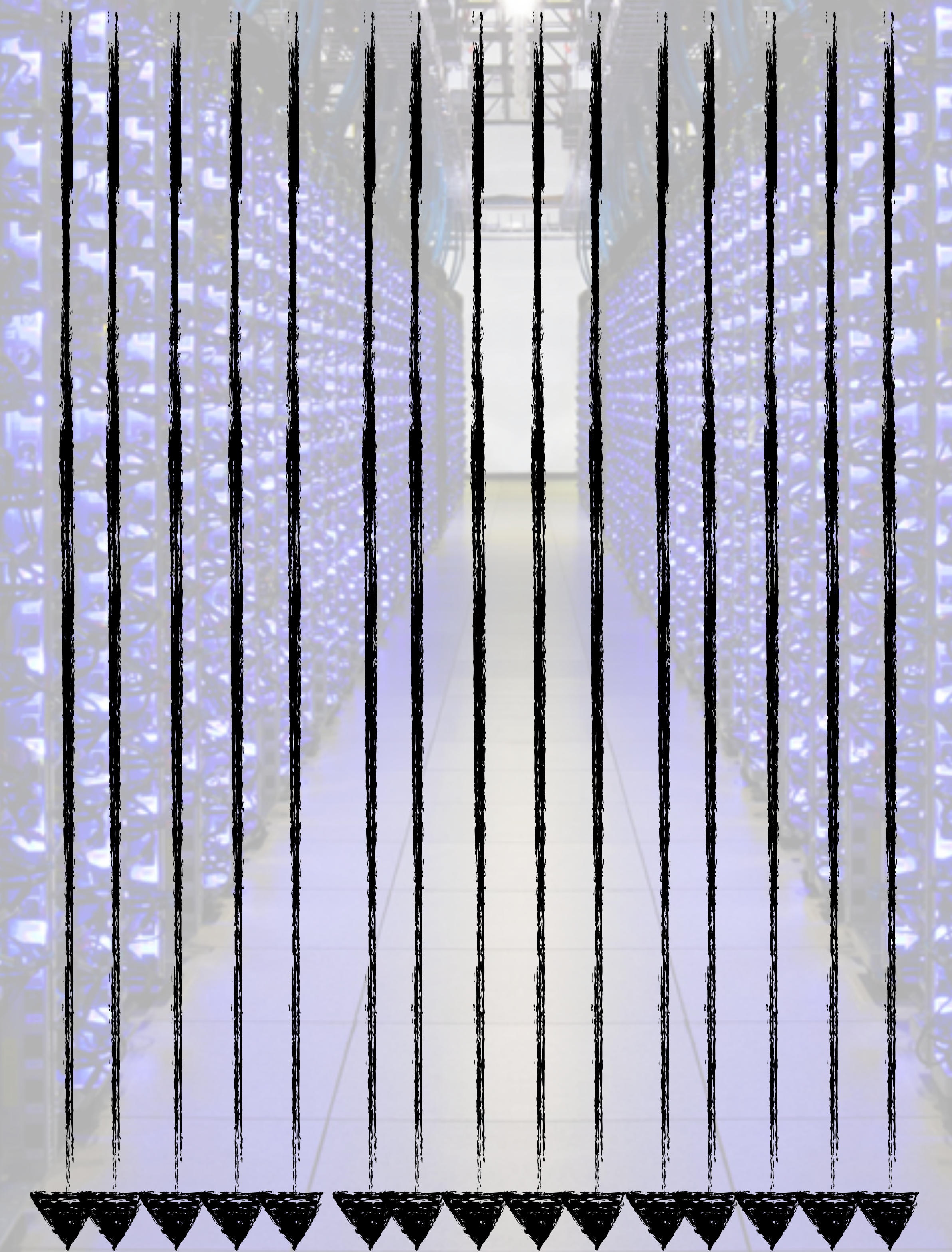






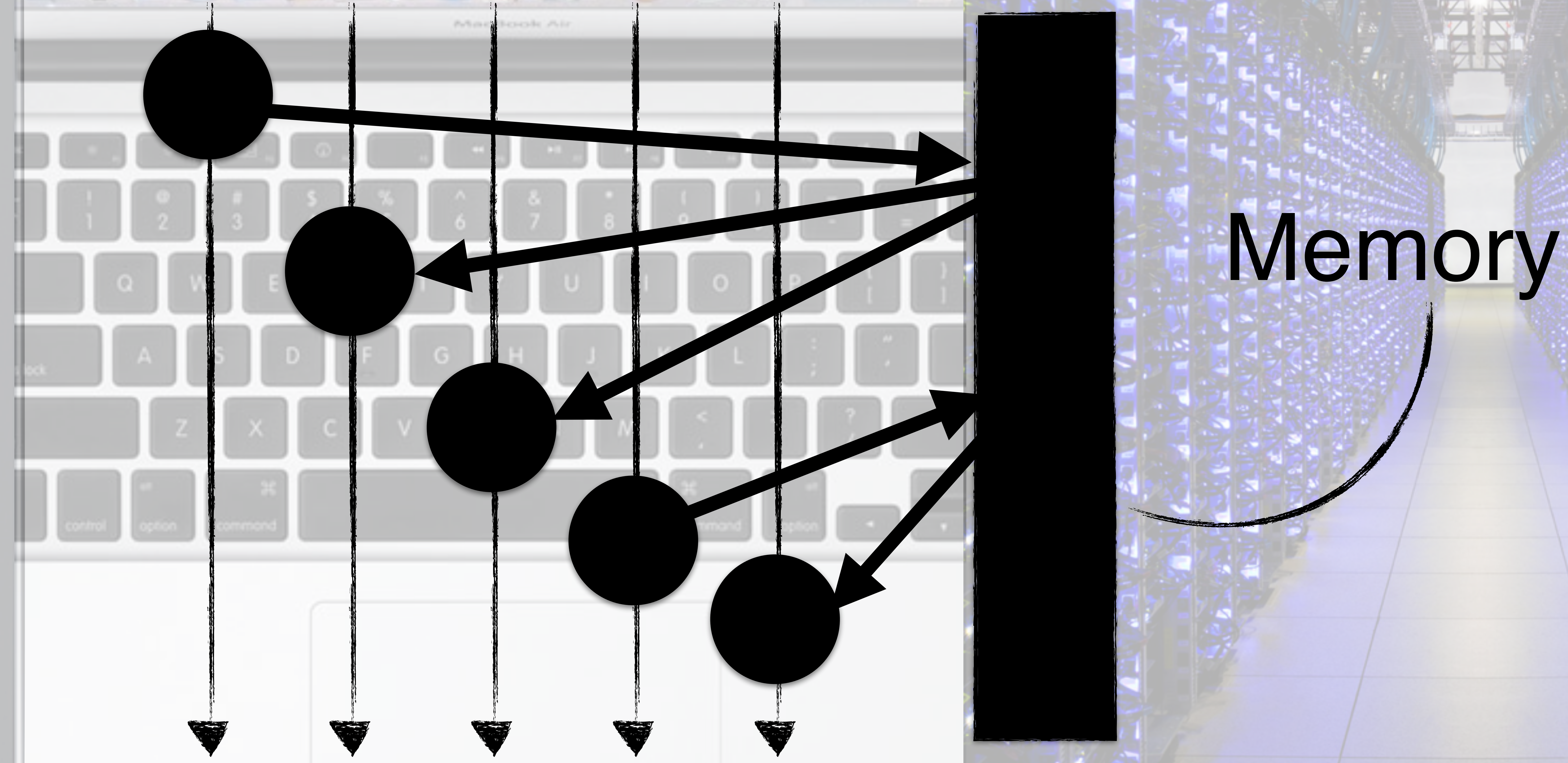
Multithreaded software is difficult to write





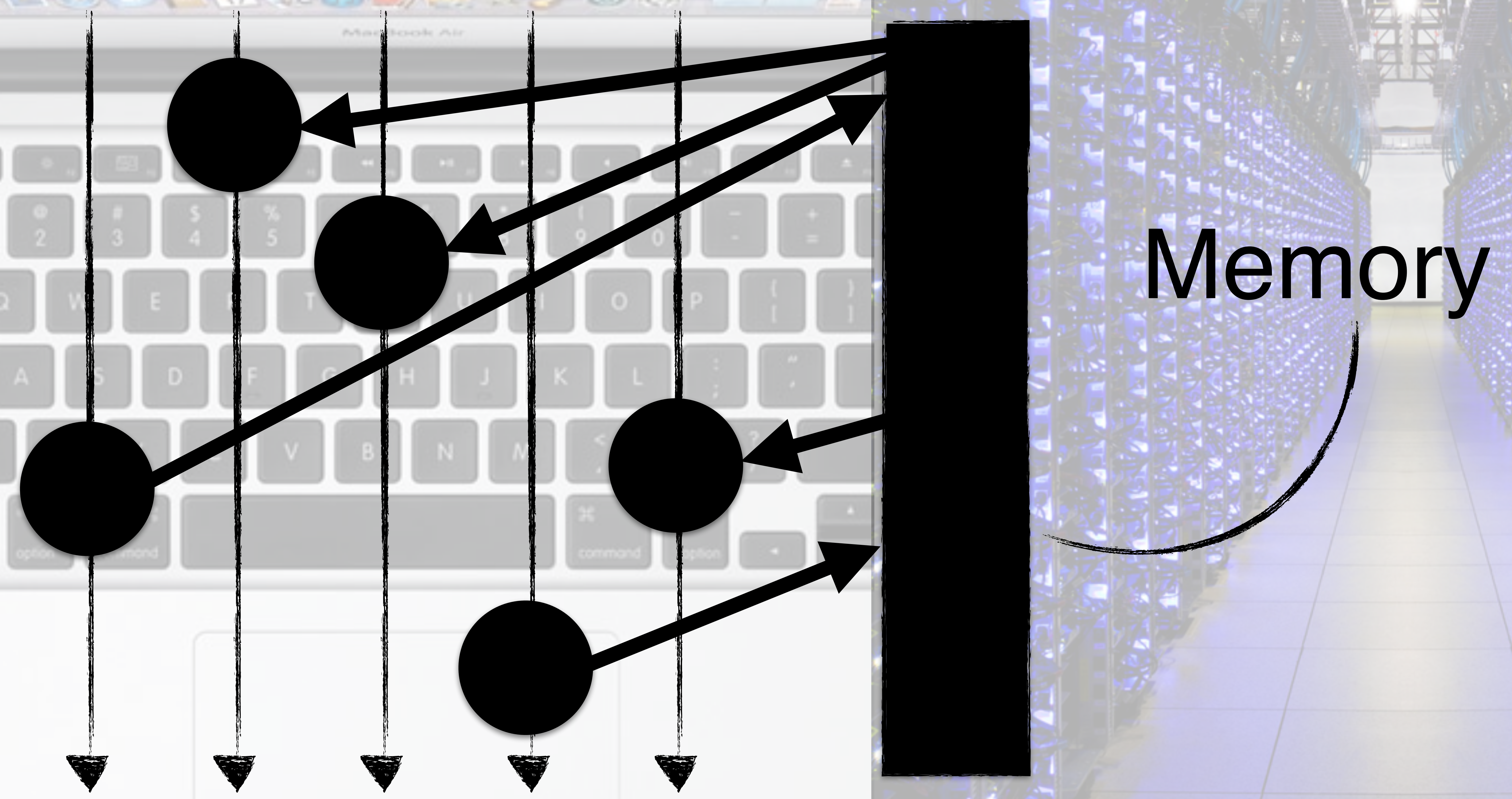
Need to think about many threads instead of one

Threads interact via shared memory

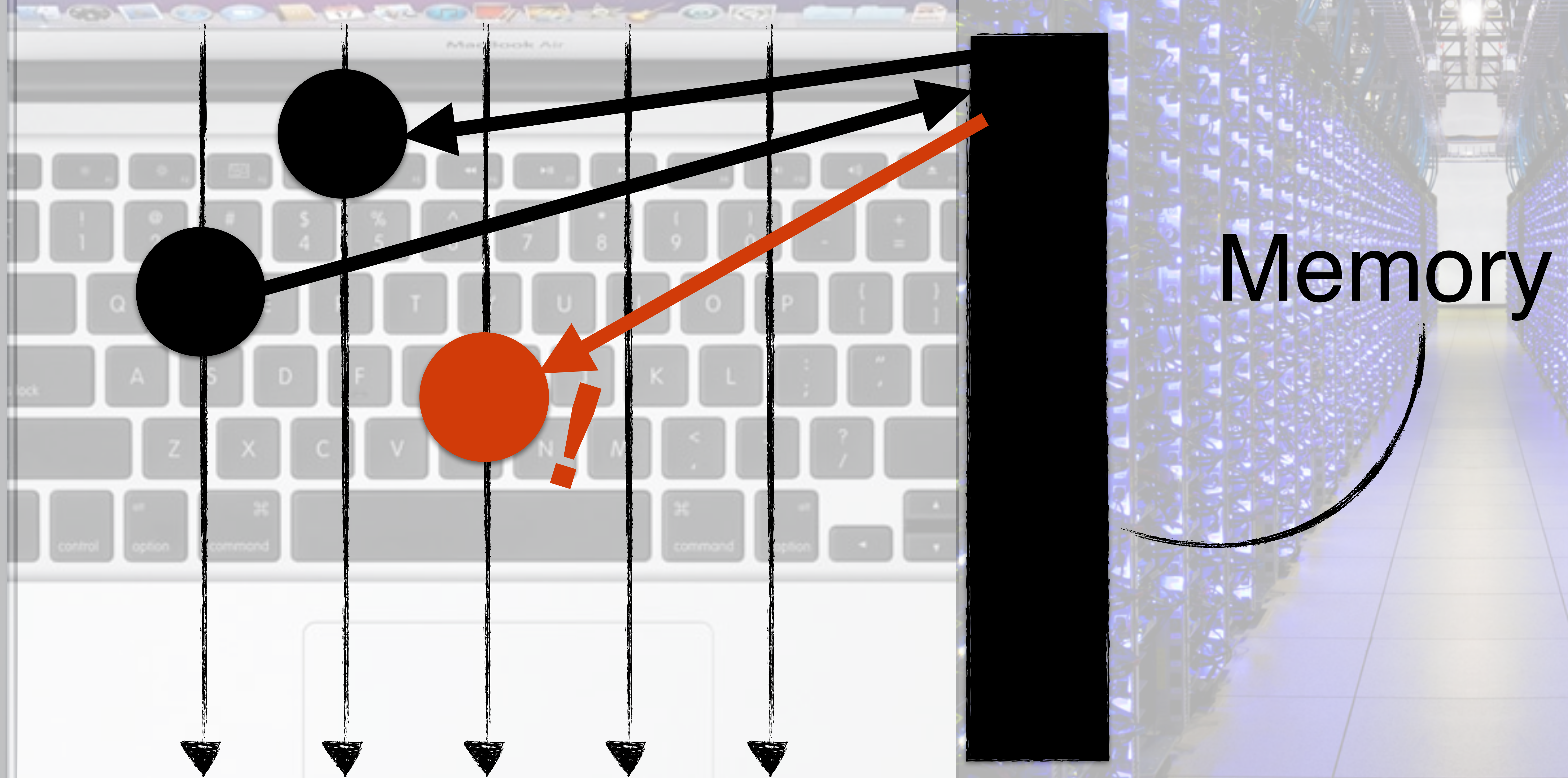


Reasoning about concurrent shared accesses is hard

Behavior can change from one execution to the next

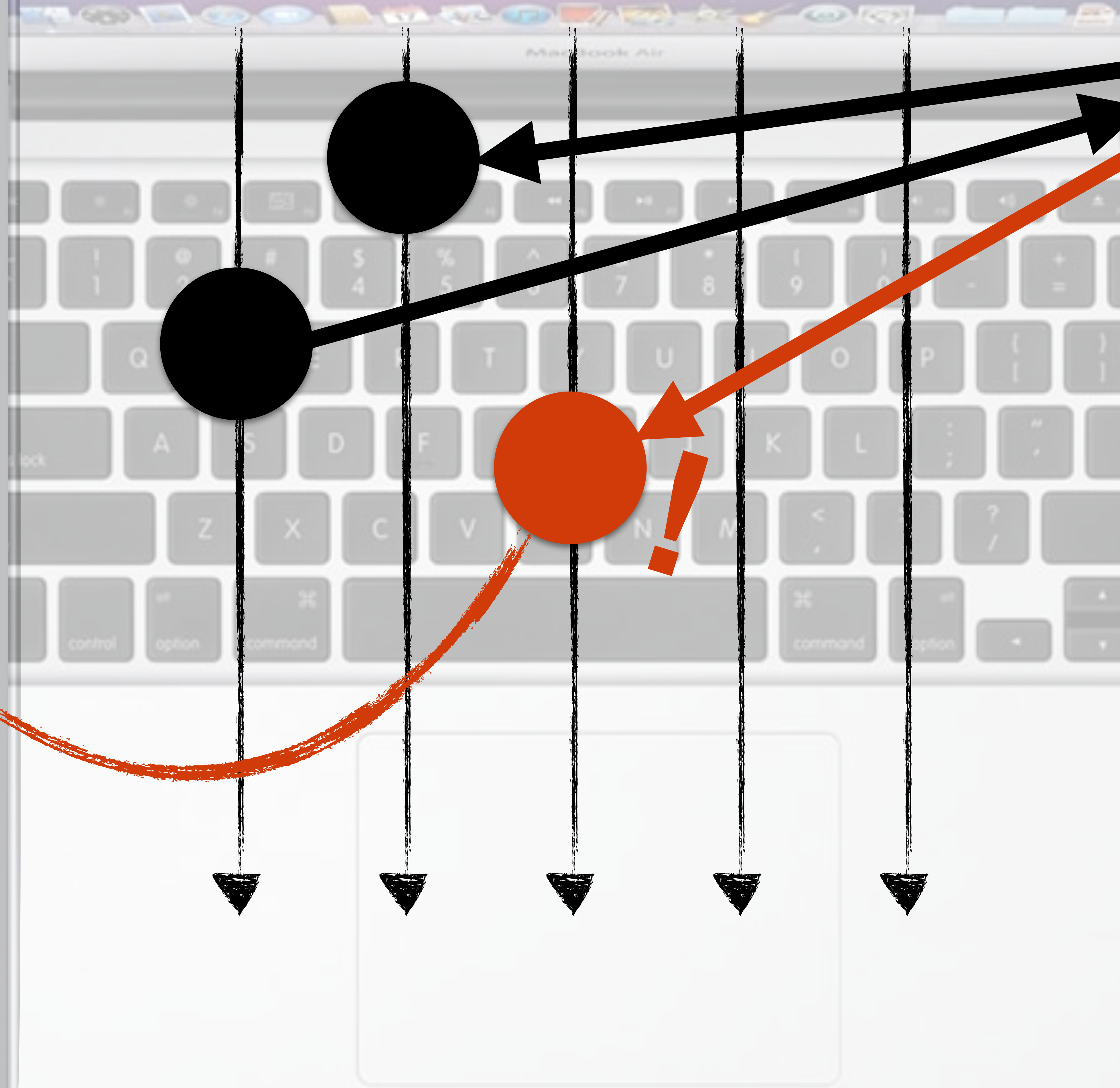


Some behavior is bad, like a **crash** or **hang**



Key problem: understanding why bad things happen

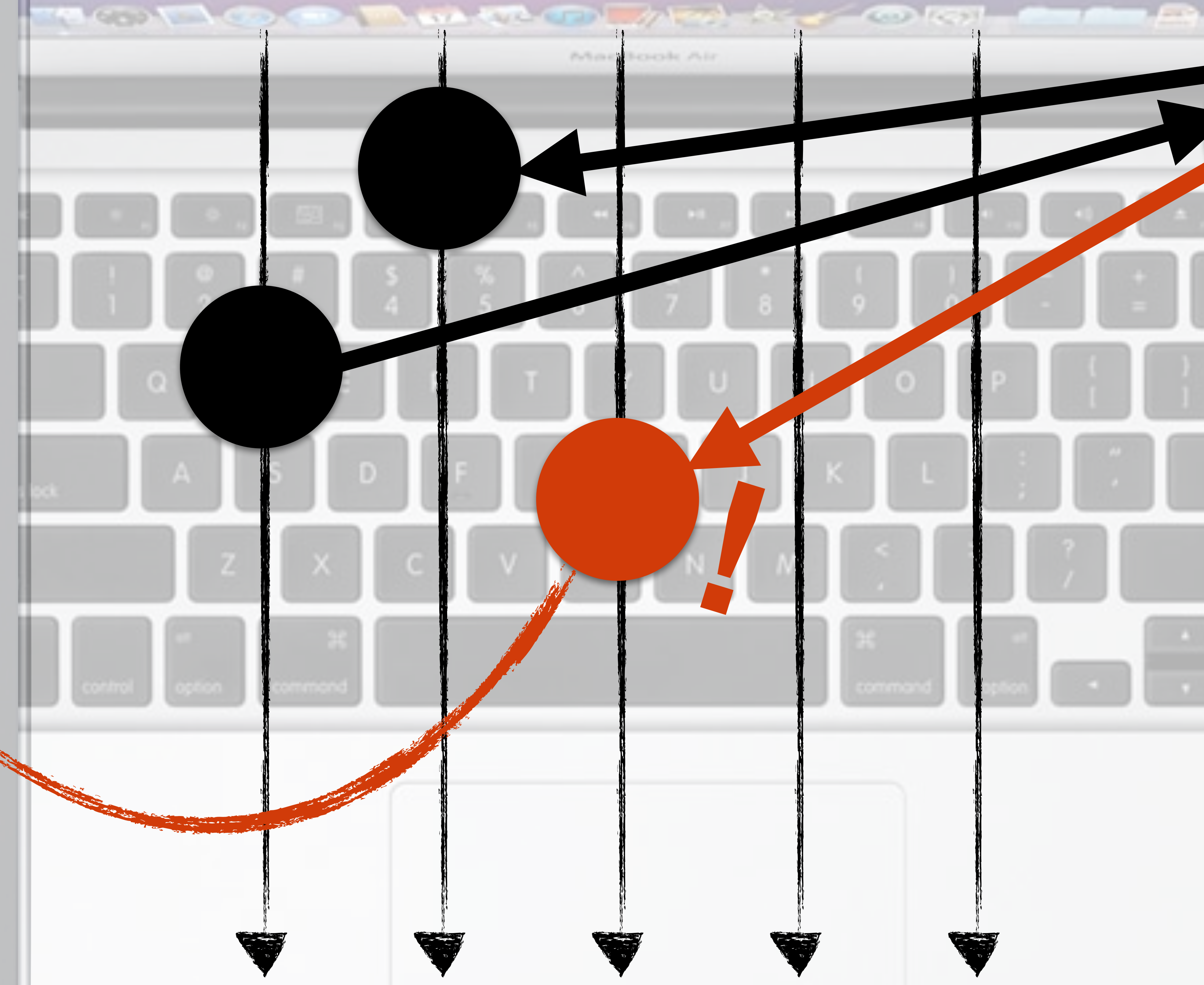
Typically debug with a **core dump**



This thread crashed @ "assert(ptr != NULL)"

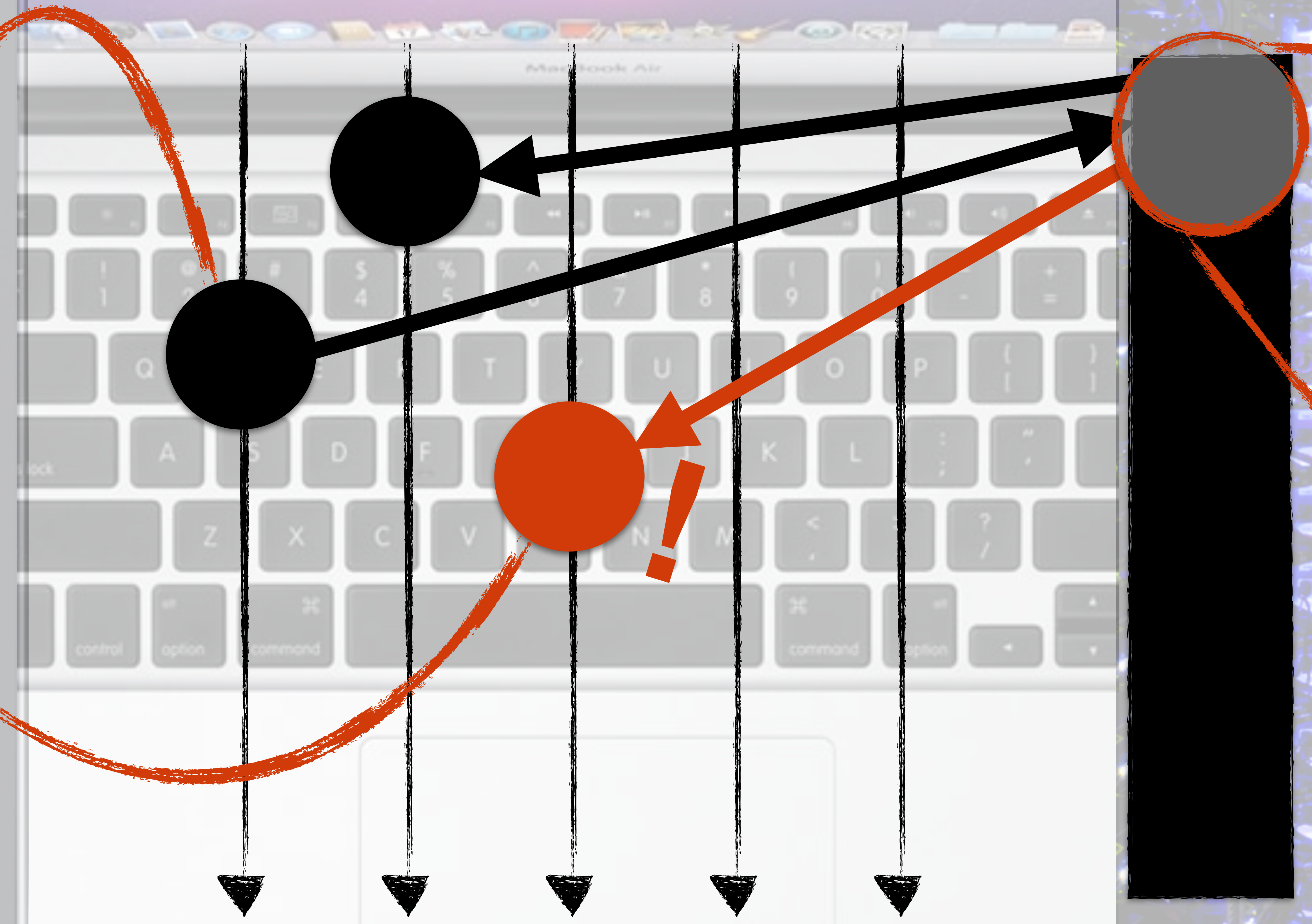
ptr = NULL

Typically debug with a **core dump**



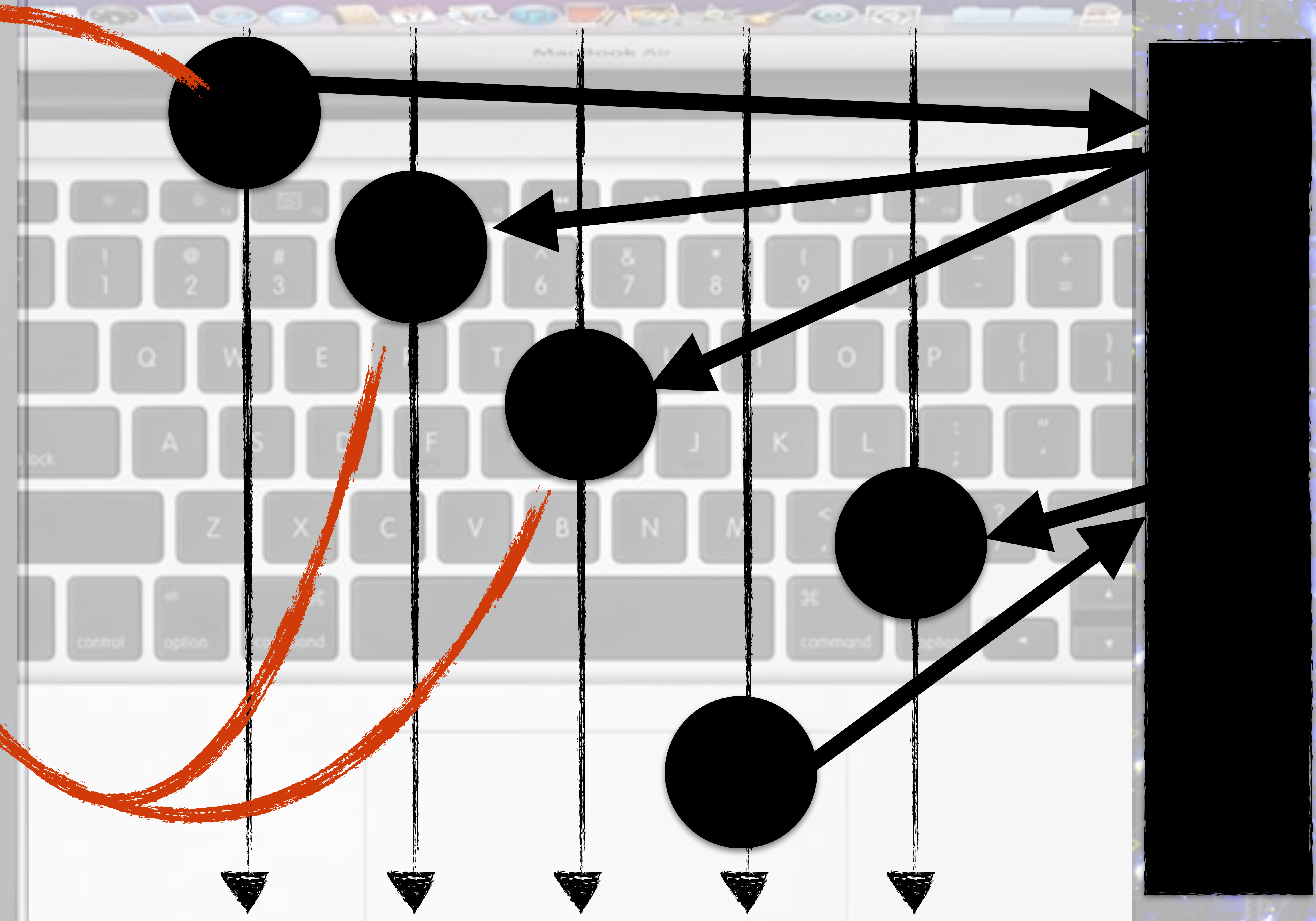
Core dump tells us **what** happened, not **why.**

Our work answers the **why?** question.



Last Writer Slices record each value's **provenance**

Bonus: provenance reveals **communication**



LWS

This thread wrote x here…

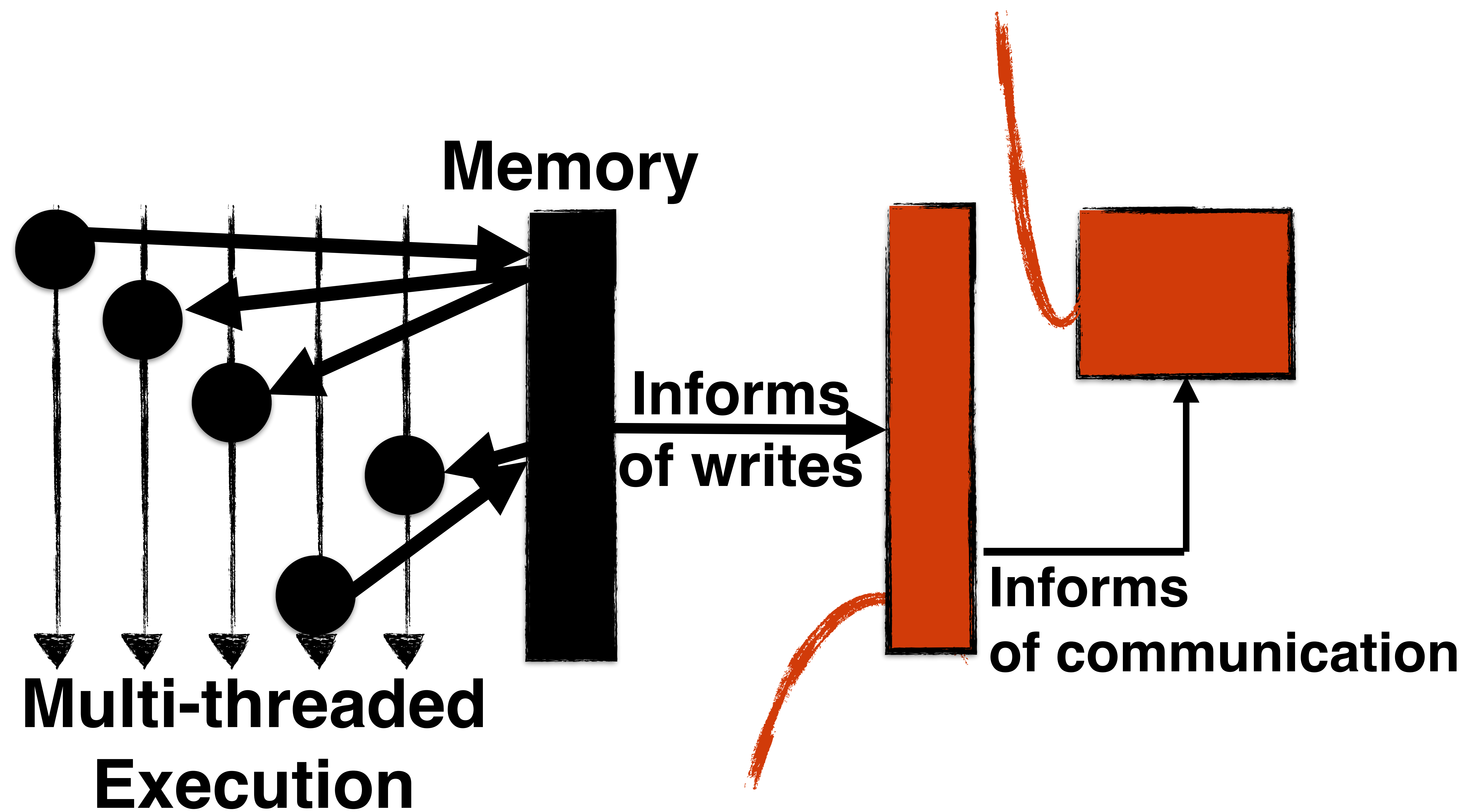
CTraps

Check LWS… Communication! Reader != last writer

```
CT_Handler(...){
  build_c_graph();
  check_atomicity();
  coop_bug_iso();
}
```

Communication Traps: custom communication handlers

CTraps Executes application-specific handlers when threads communicate



LWS Tracks data provenance at runtime with low overhead

Debugging

Programmer examines provenance via LWS

Analysis

Arbitrary concurrency analyses via CTraps

Efficiency

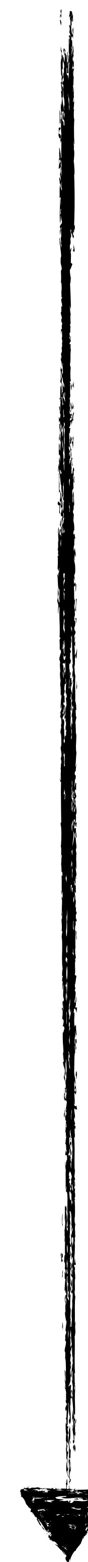
Overheads low enough for production use

append()

Shared Variables

len:
length of string

str:
string buffer

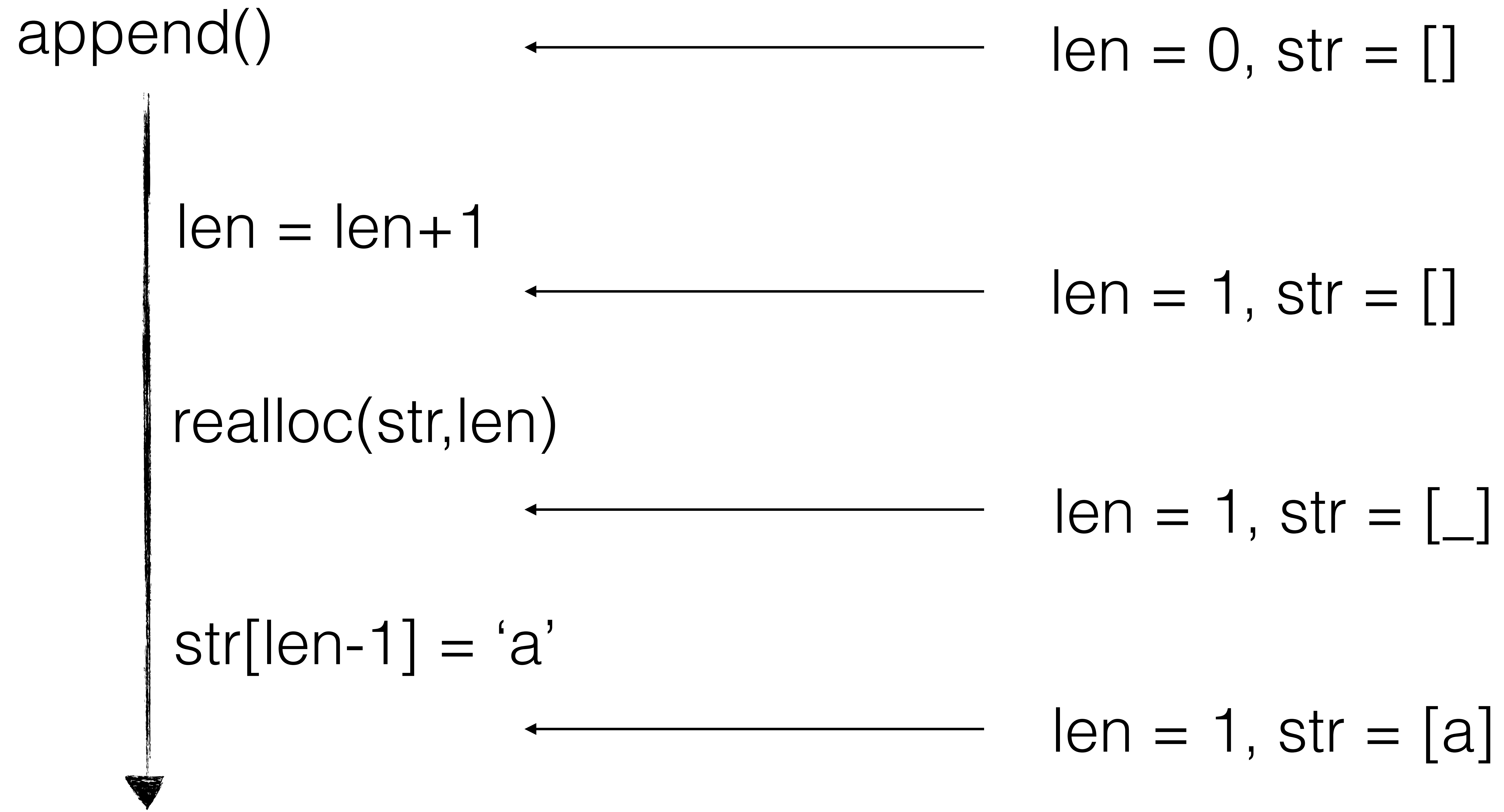


```
len = len+1

realloc(str,len)

str[len-1] = 'a'
```

Shared Variables

len:
length of string

str:
string buffer



append()

len = len+1

realloc(str,len)

str[len-1] = ‘a’ **Crash: str[len-1] out of bounds**

Shared Variables

len:
length of string

str:
string buffer

Shared Variables

len:
length of string

str:
string buffer

append()

Programmer: “This must be wrong”

```
len = len+1

realloc(str,len)

str[len-1] = 'a'
```

! Crash: str[len-1] out of bounds

Programmer: "One of these must be wrong"

Shared Variables

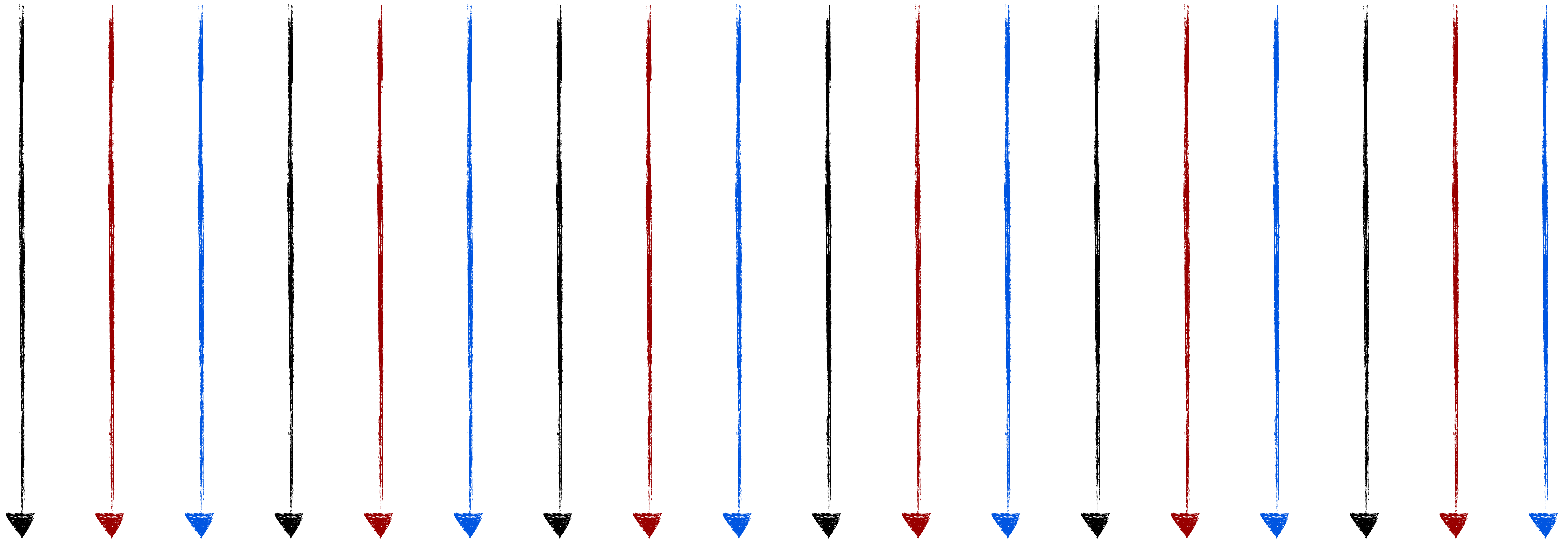
len:
length of string

str:
string buffer

append()	append()	erase()
len = len+1	len = len+1	len = len -1
realloc(str,len)	realloc(str,len)	
str[len-1] = 'a' !	str[len-1] = 'a'	

Electrical & Computer ENGINEERING

Programmer: “WTF?!”



append()

len = len+1

realloc(str,len)

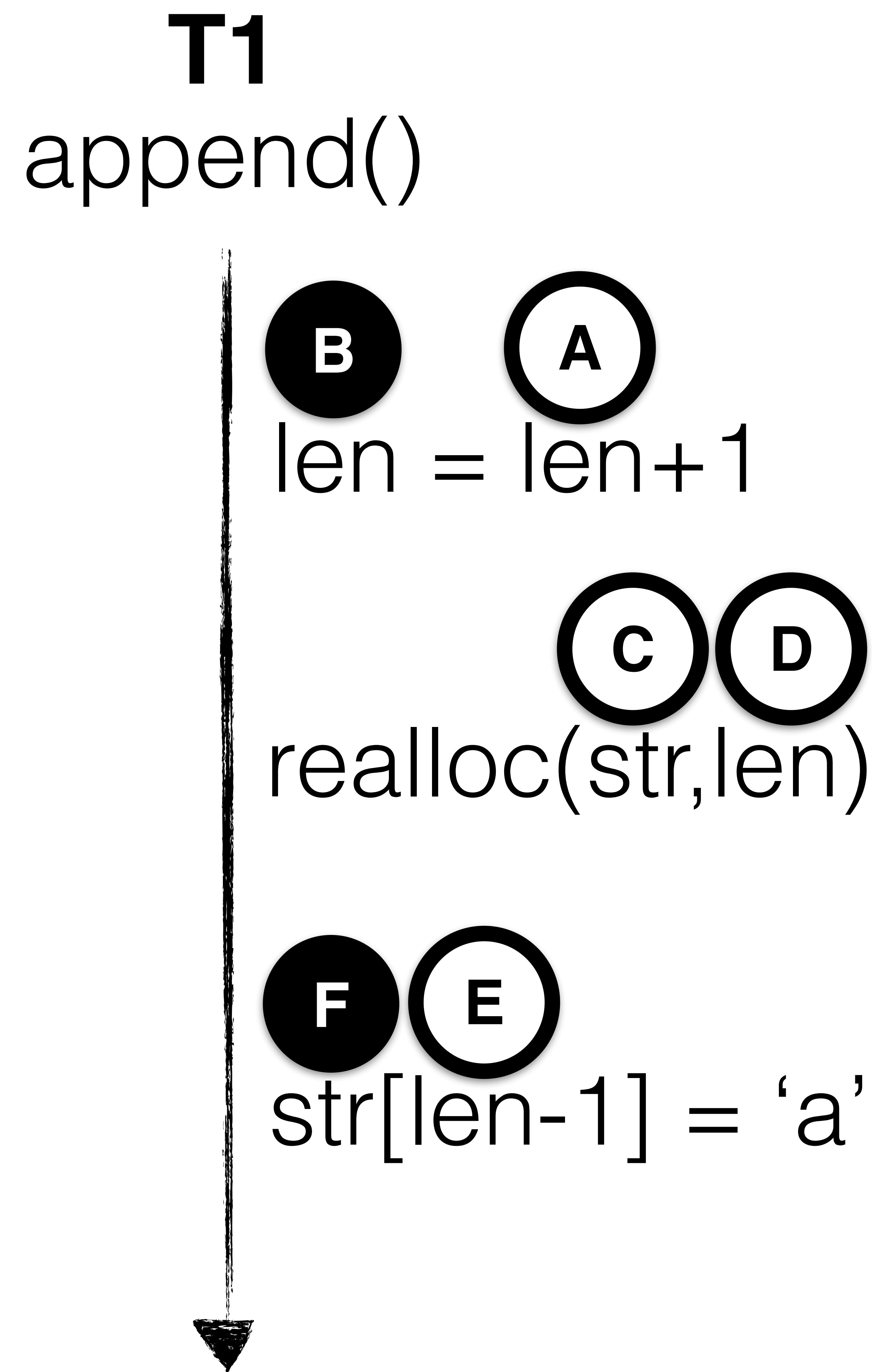
str[len-1] = ‘a’

Shared Variables

len:
length of string

str:
string buffer

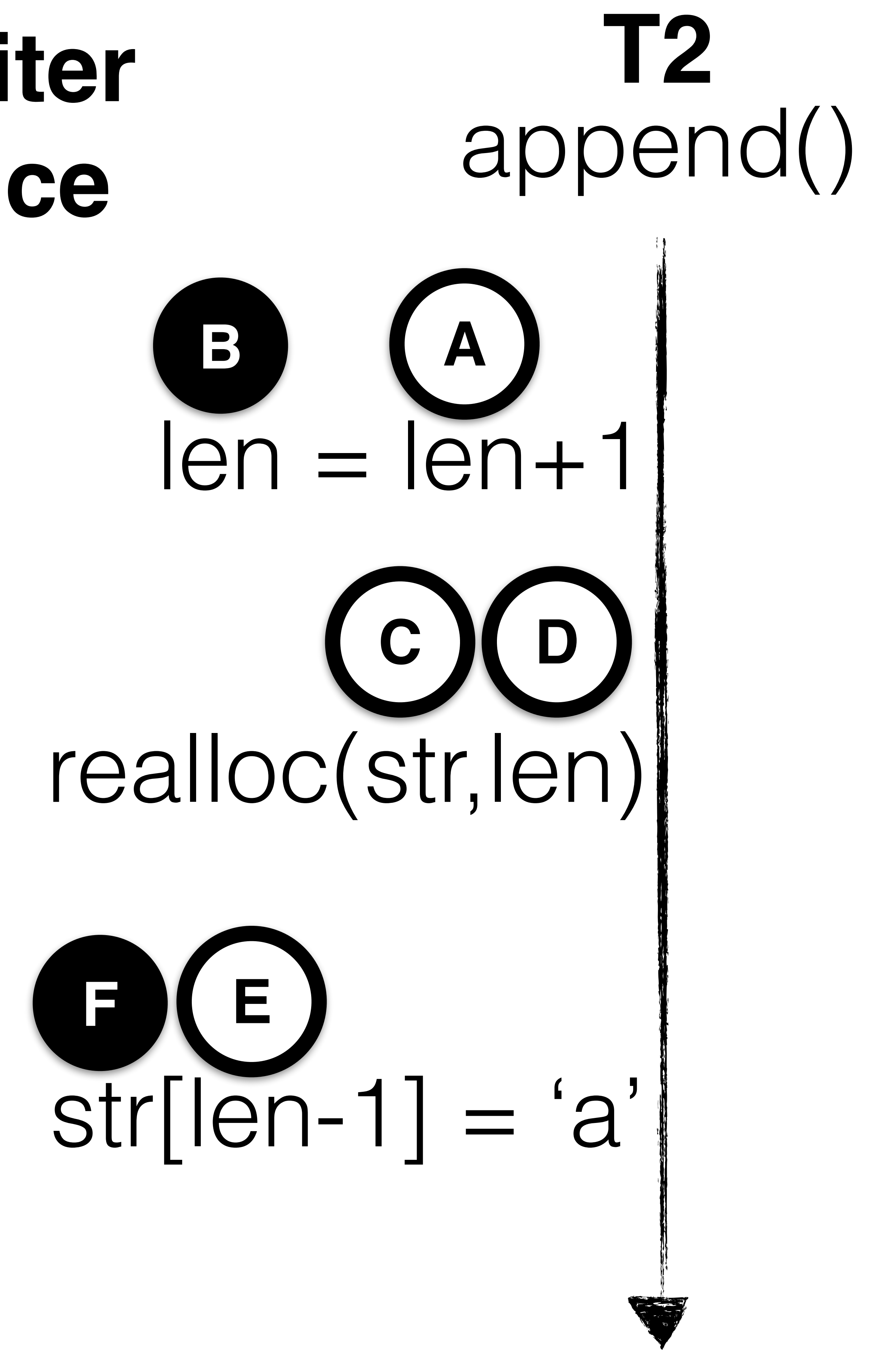
Last Writer Slices tracks **data provenance**: **thread** & **code point** that last wrote **len**



Last Writer Table

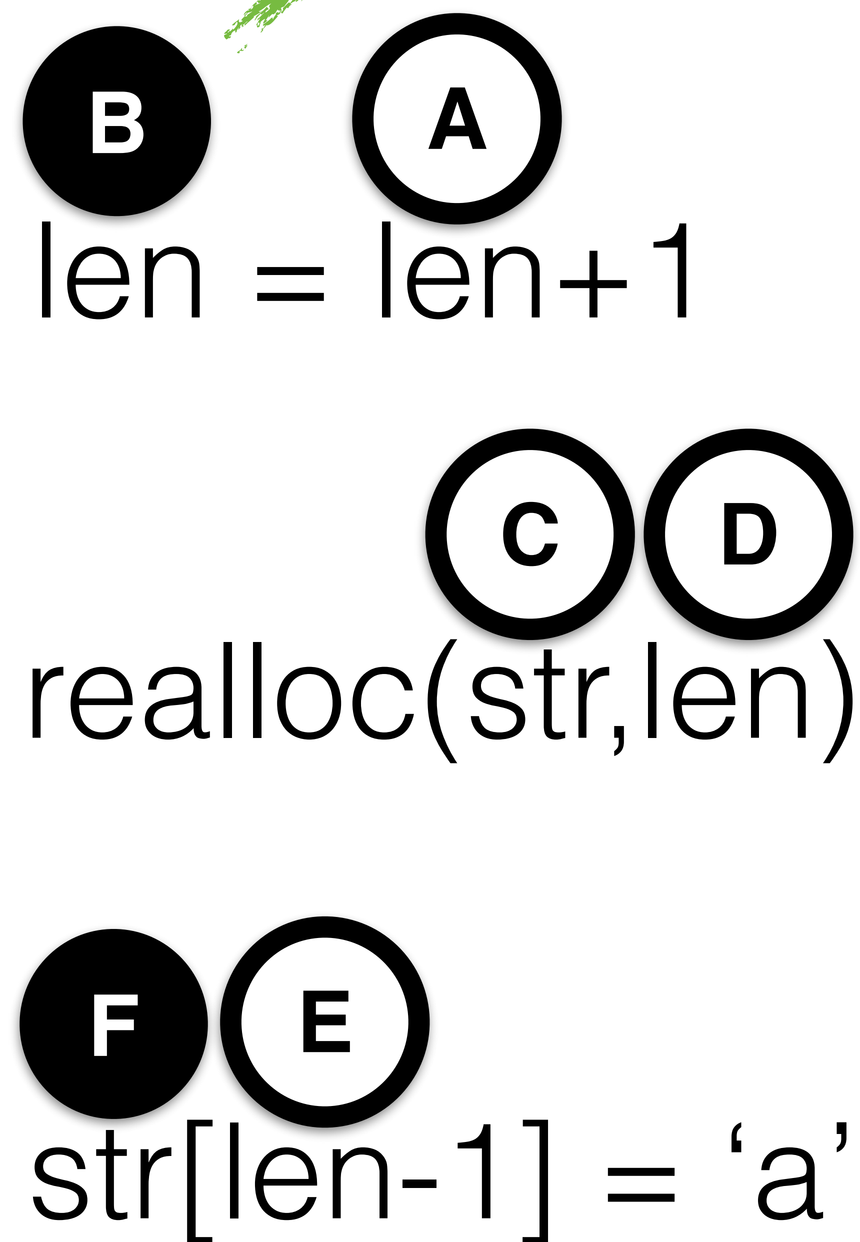
Var	Thread	Code Pt.

Last Writer Slice



X **Read Operation**

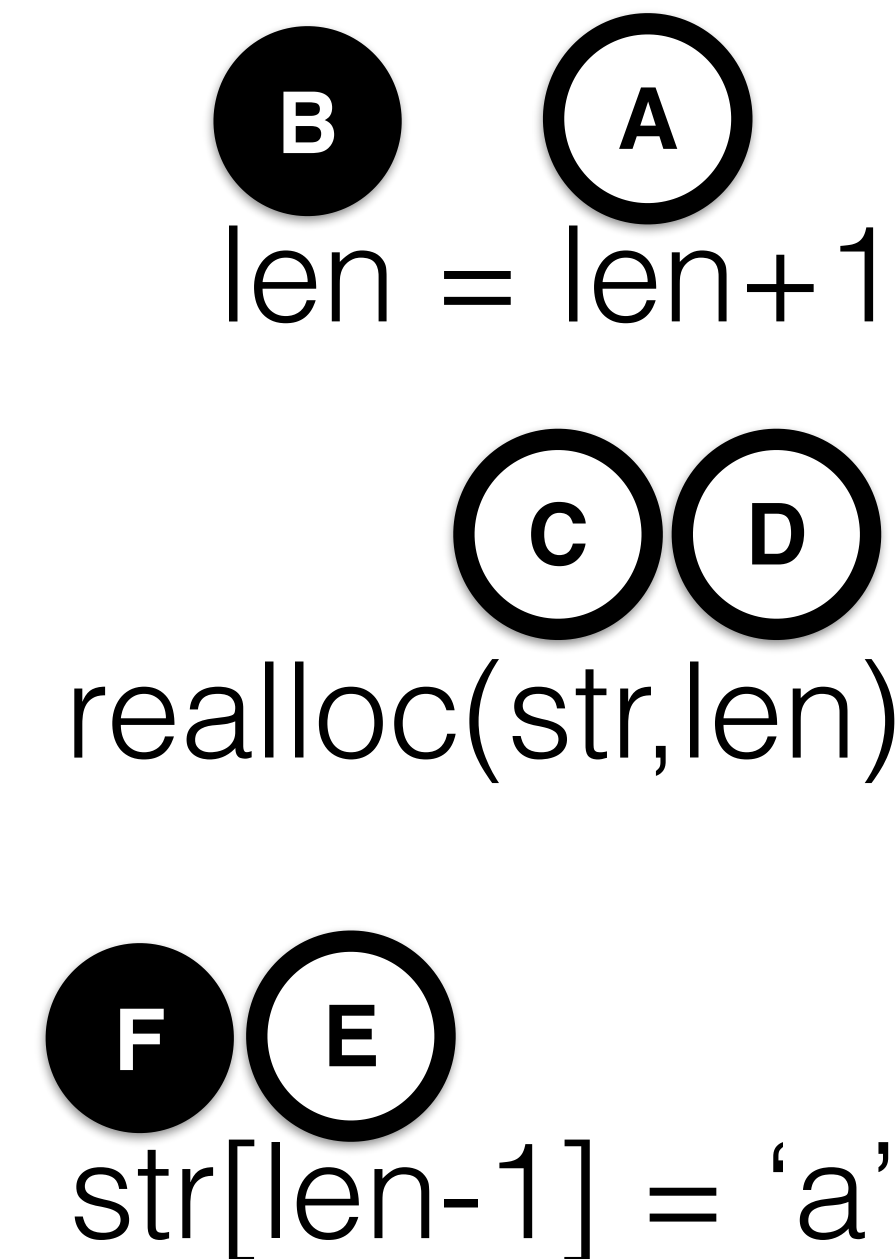
Y **Write Operation**



Update

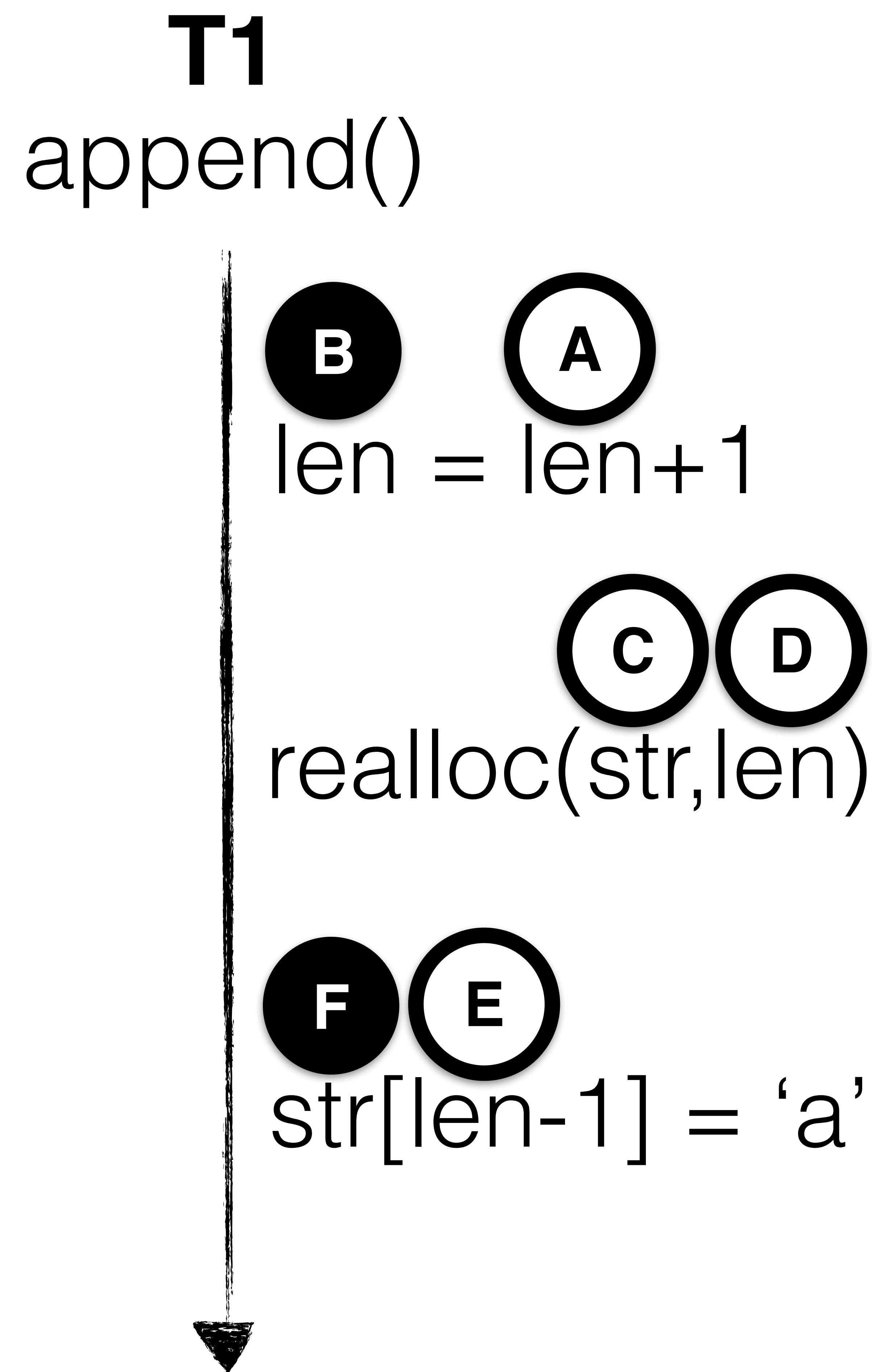
Last Writer Table

len	T1	B



X **Read Operation**

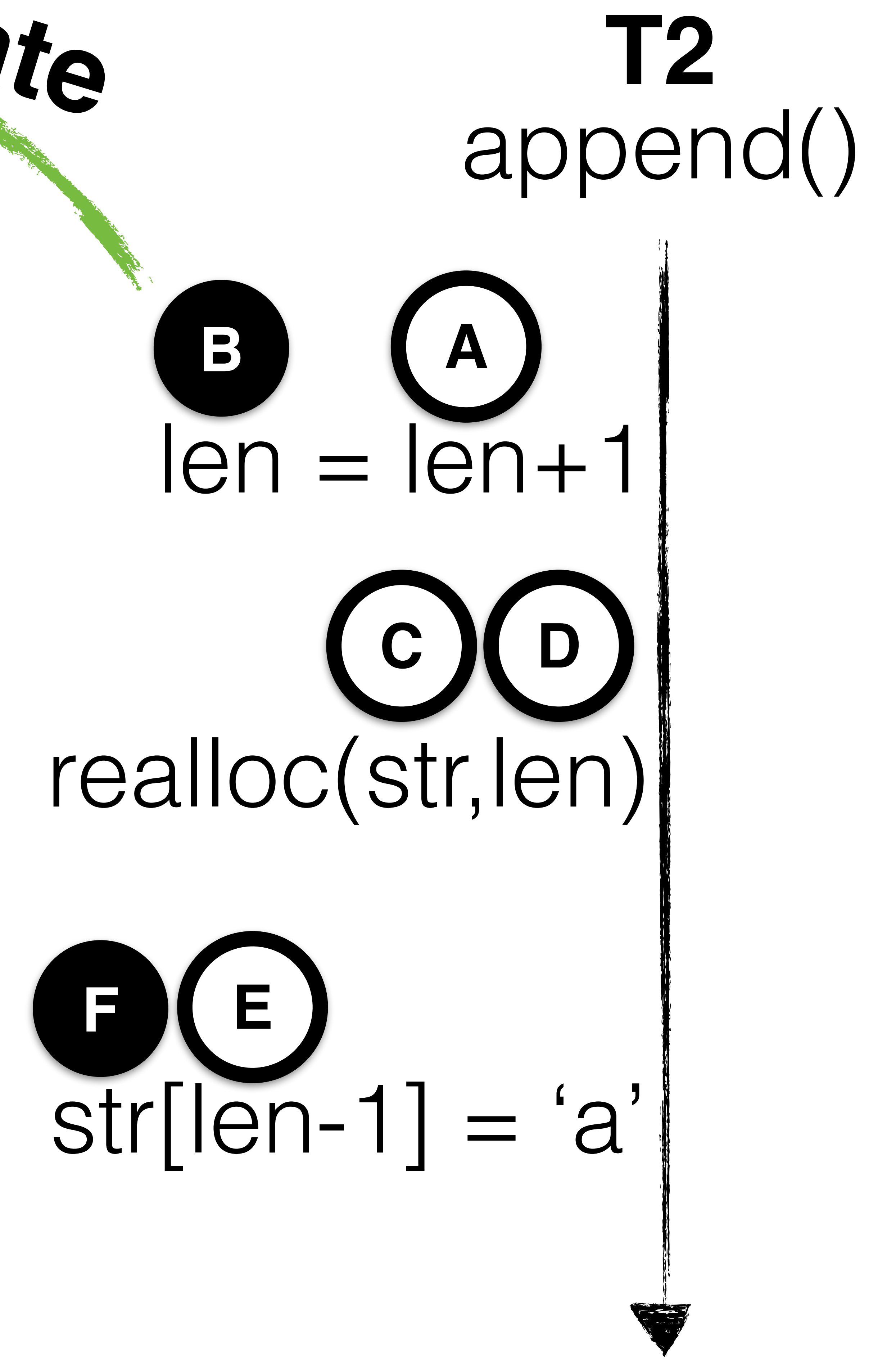
Y **Write Operation**



Last Writer Table

len	T2	B

Update



X **Read Operation**

Y **Write Operation**

Last Writer Table

len	T2	B

T1
append()

B A
len = len+1

C D
realloc(str,len)

F E
str[len-1] = 'a'

!Crash

T2
append()

B A
len = len+1

C D
realloc(str,len)

F E
str[len-1] = 'a'

X **Read Operation**

Y **Write Operation**

Last Writer Table

len	T2	B

T1
append()

B A
len = len+1

C D
realloc(str,len)

F E
str[len-1] = 'a'

Breakpoint

T2
append()

B A
len = len+1

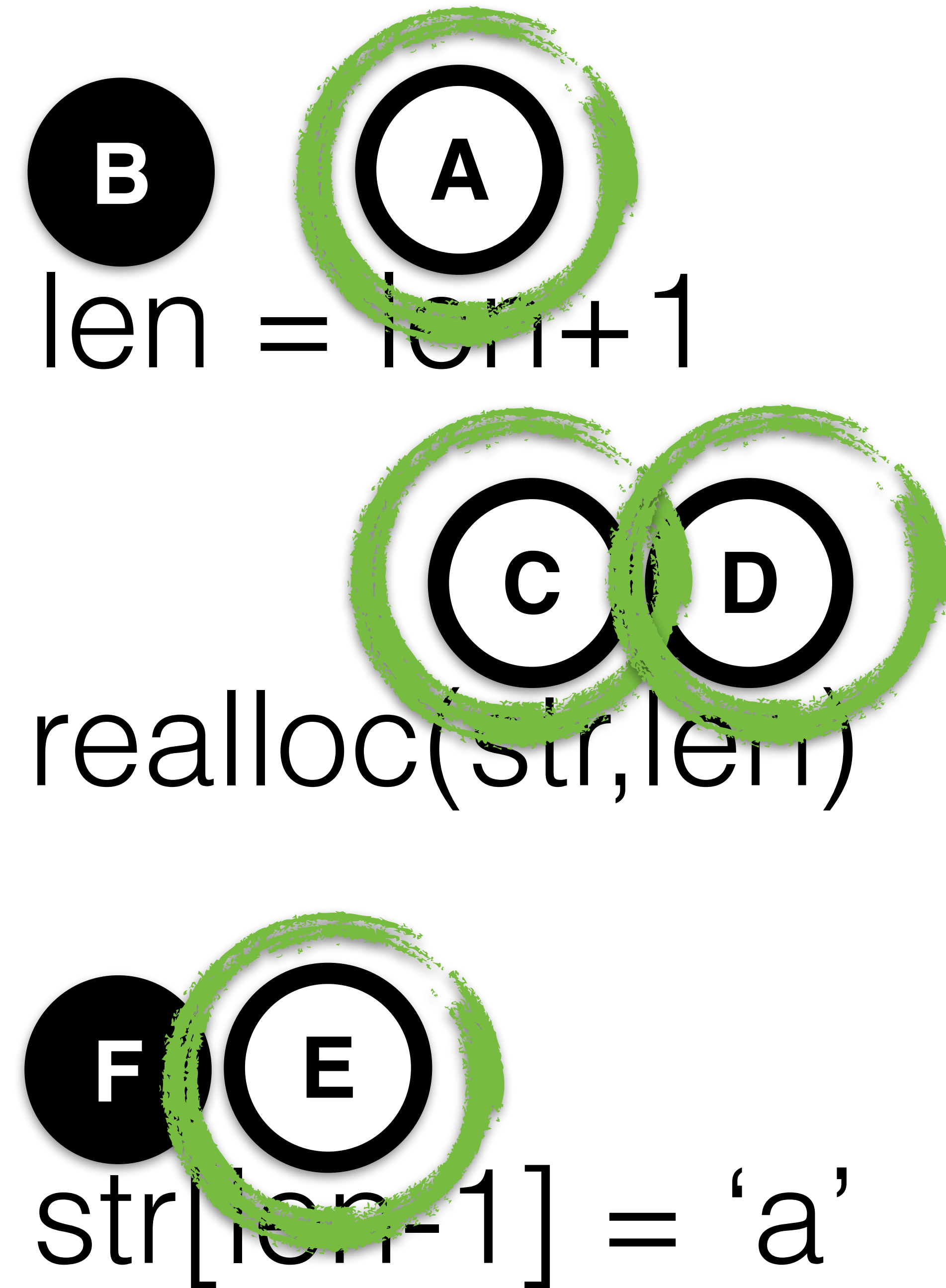
C D
realloc(str,len)

F E
str[len-1] = 'a'

X **Read Operation**

Y **Write Operation**

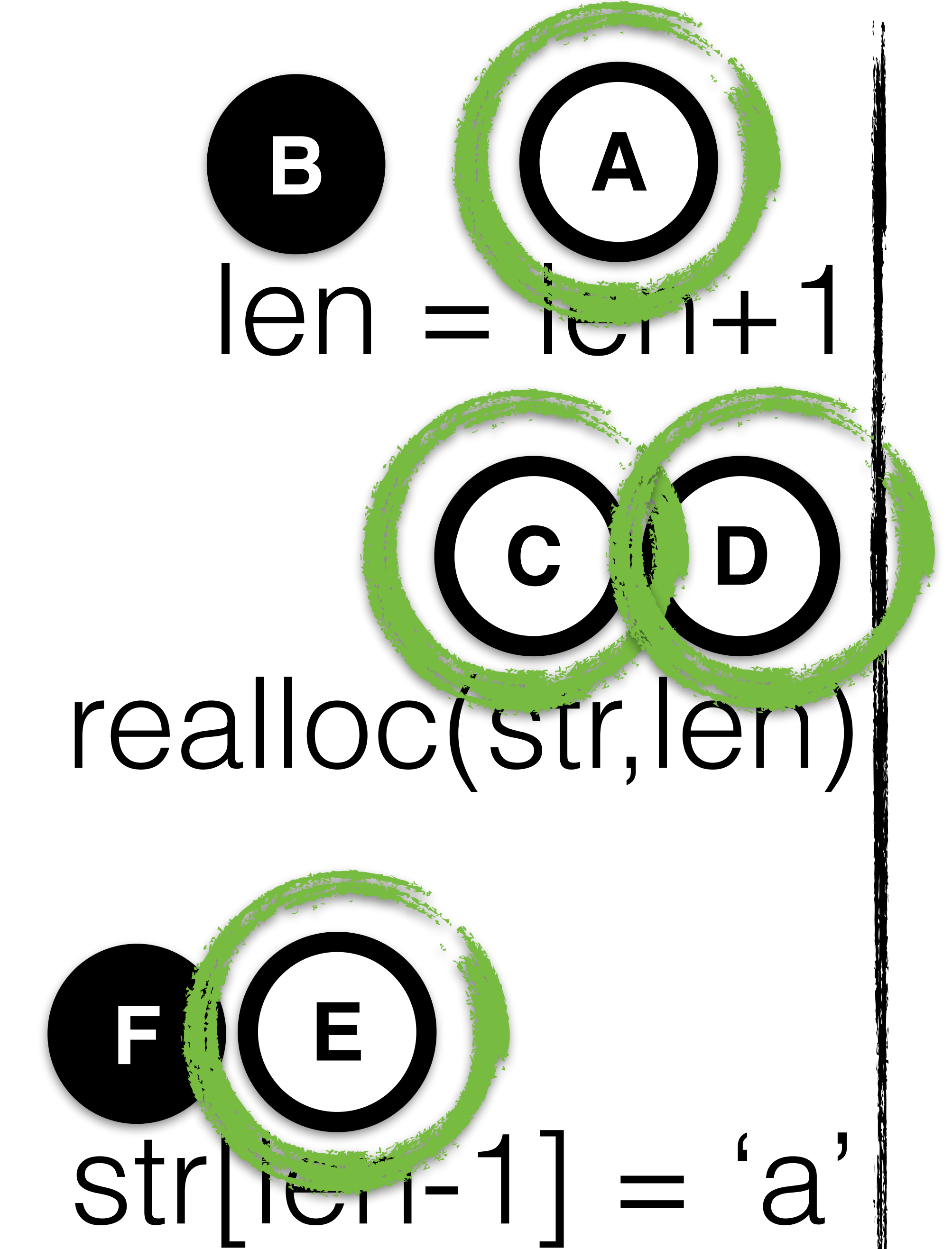
T1
append()



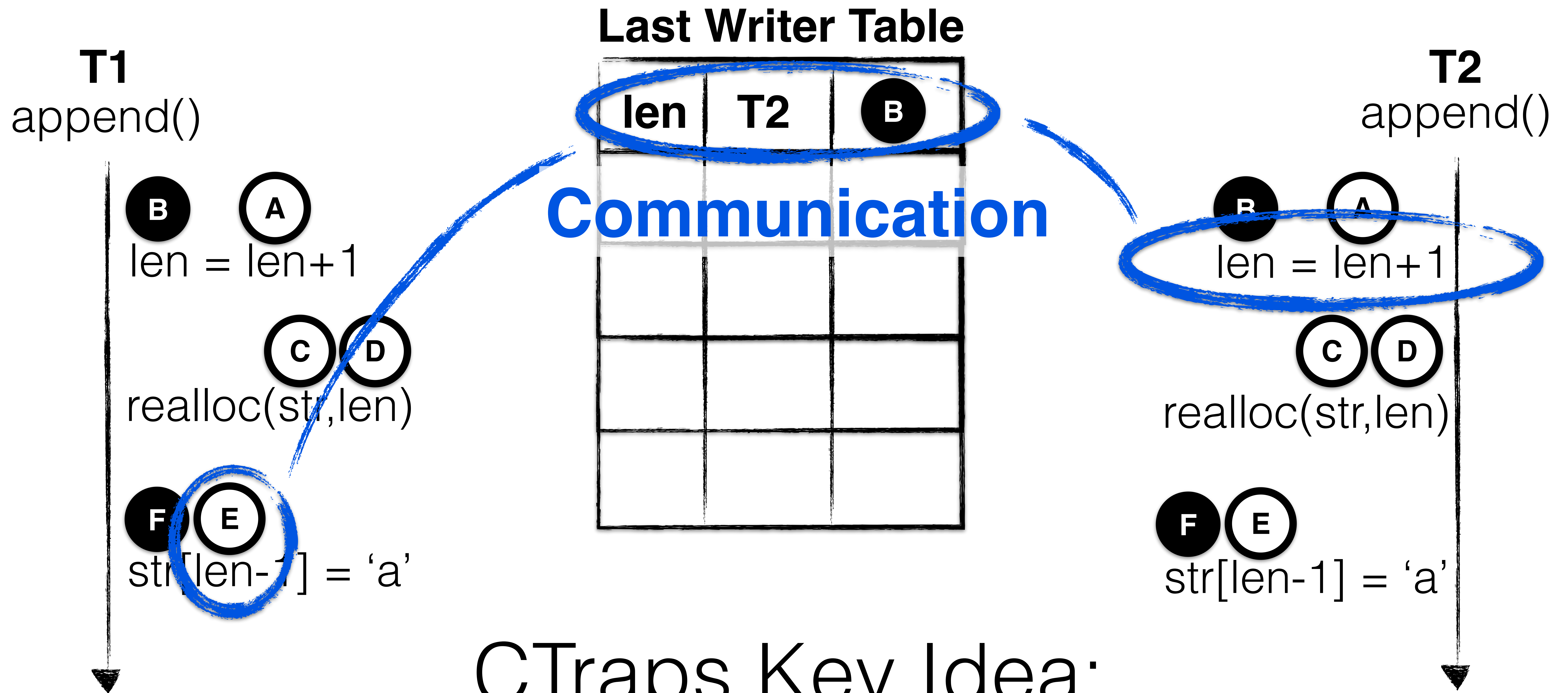
Last Writer Table

len	T2	B

T2
append()



Reads are **free** for LWS

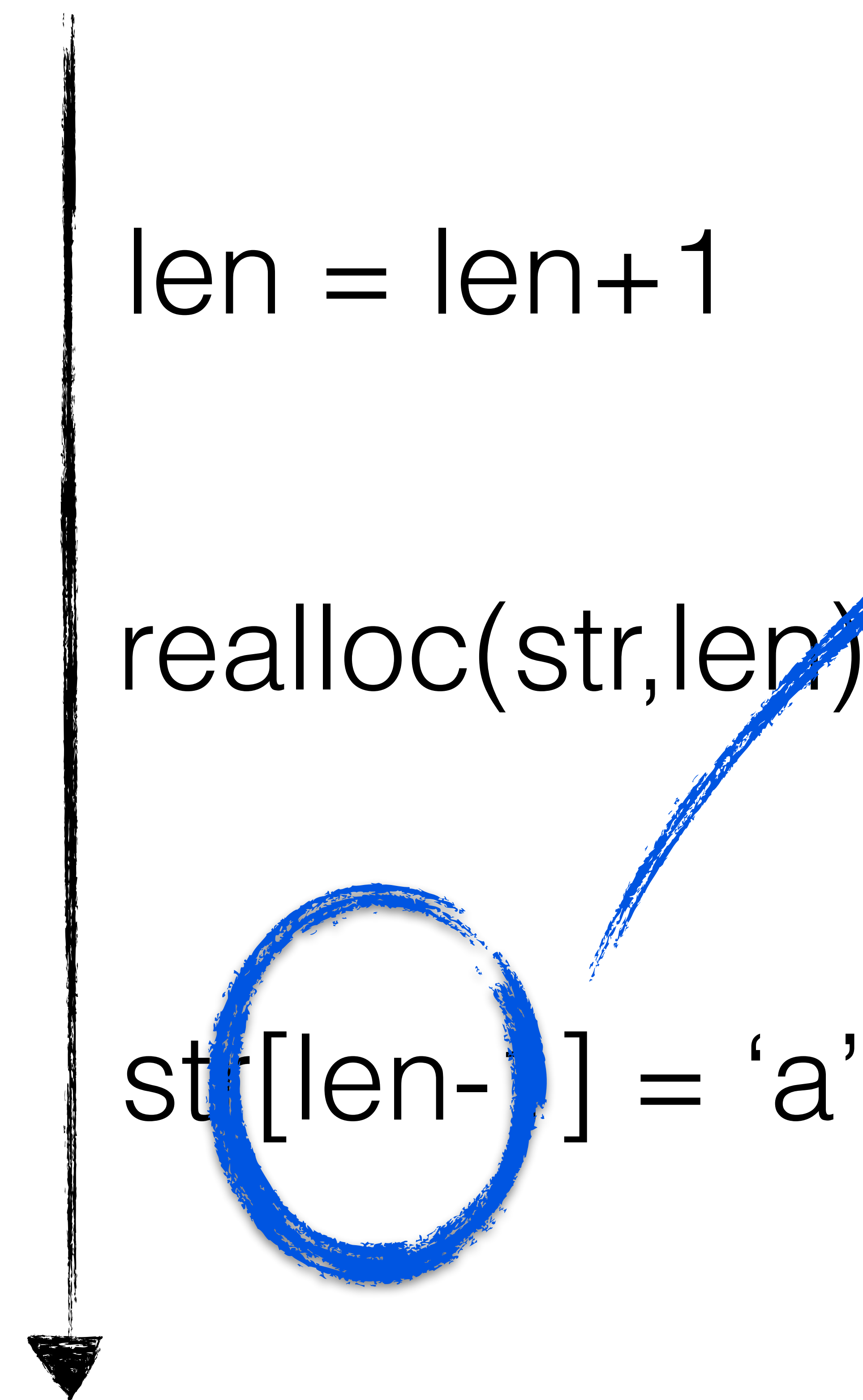


CTraps Key Idea:

Different thread in the LWT? Threads are **communicating.**

CTraps allows **communication handlers**

append()

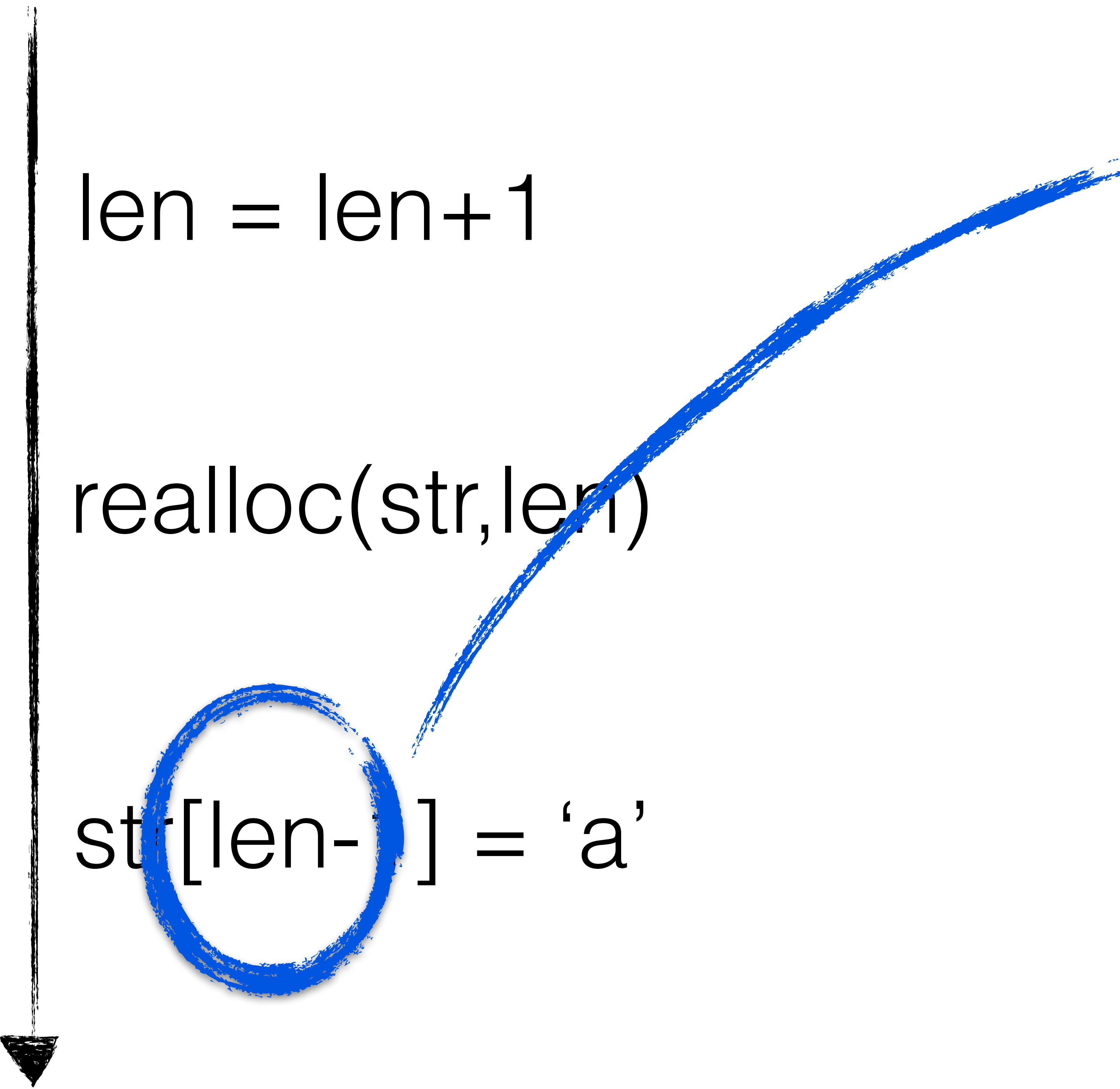


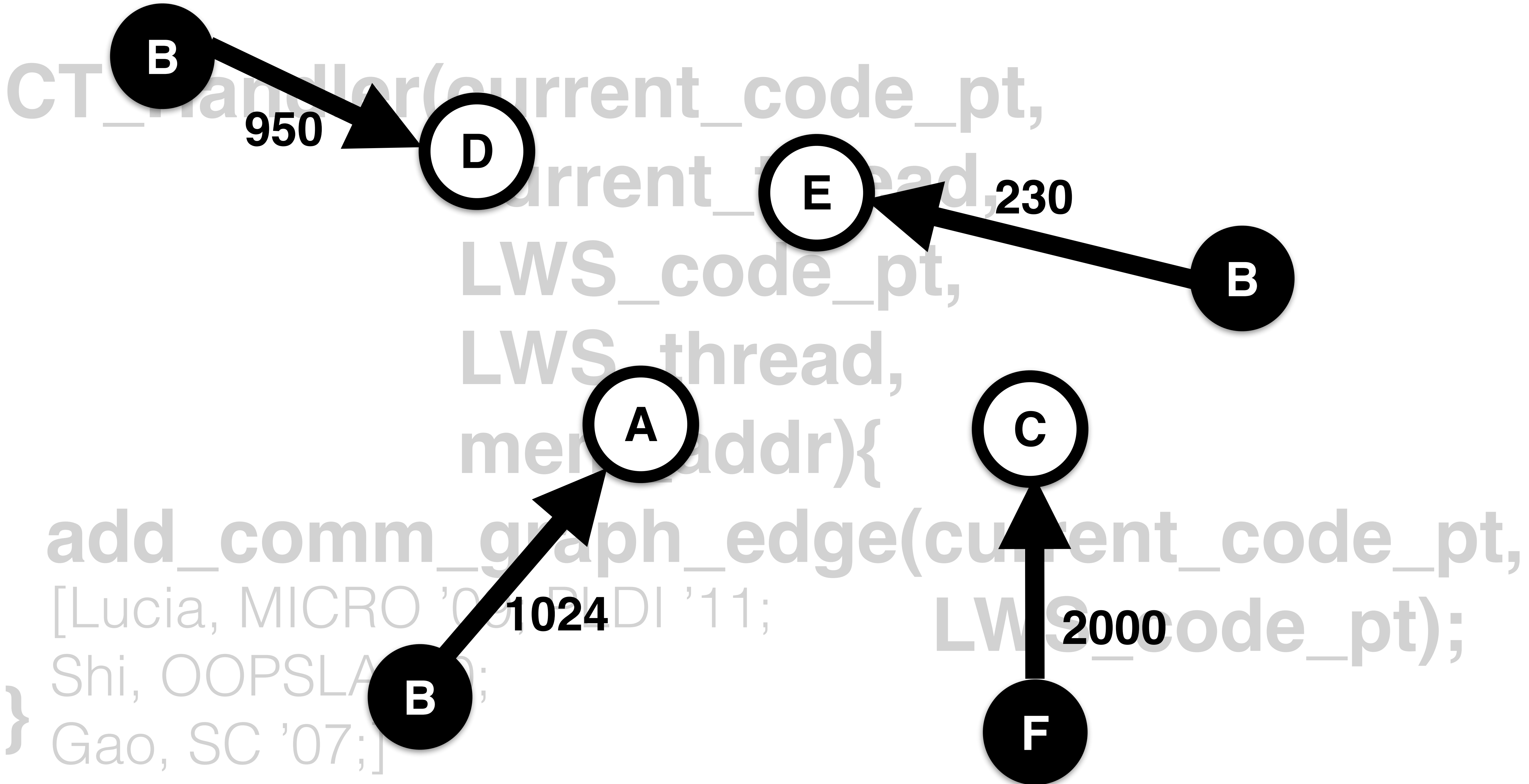

```
CT_Handler(current_code_pt,
           current_thread,
           LWS_code_pt,
           LWS_thread,
           mem_addr){
  add_comm_graph_edge(current_code_pt,
                      LWS_code_pt);
}
```

[Lucia, MICRO ’09; PLDI ’11; Shi, OOPSLA ’10; Gao, SC ’07;]

Handlers implement arbitrary **communication analysis**

CTraps allows **communication handlers**





Handlers implement arbitrary **communication analysis**

Last Writer Slicing & CTraps Implementation

LWS & CTraps Compiler

Link Program to Runtimes

@ Write:
update LWT;
call CTraps Handler

@ Read:
call CTraps Handler

LWS Runtime Library

Last Writer Table

len	T2	B
str	T17	A
foo	T9	C
bar	T6	D

LWT

CTraps Runtime Library

c_graph()

Maintain List of CT_Handlers

Call CT_handlers on comm.

Expose Comm. to CTraps

Send LWS + Core Dump to debugger (GDB)

Caveat: LWS Implementation & Data-races

LWT adds no synchronization

```
Update_LWT(len)
len = len+1

realloc(str,len)

Update_LWT(str)
str[len-1] = 'a'
```

Program synchronization keeps LWT consistent

```
Update_LWT(len)
len = len+1
Release()
```

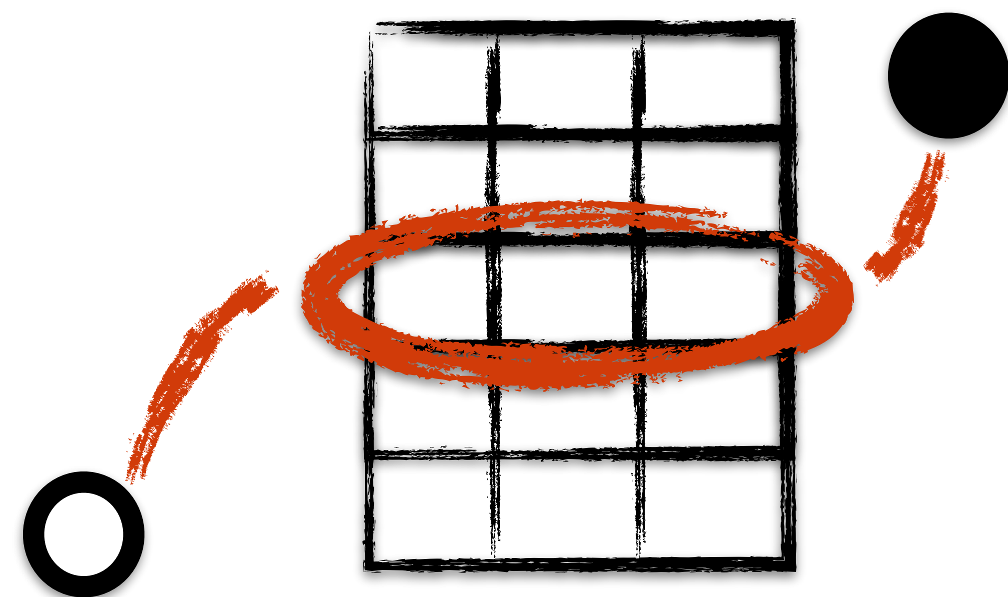
ordered!

```
Lock()
Update_LWT(len)
len = len+1
```

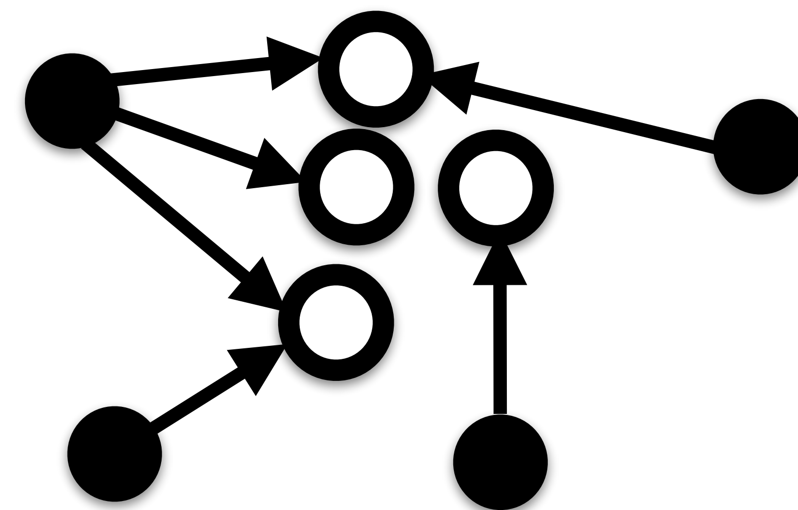
Correct for DRF programs

(may be incorrect for racy programs)

LWS helps with
Debugging



CTraps enables useful
Analysis



LWS & CTraps have
Efficiency
sufficient for production

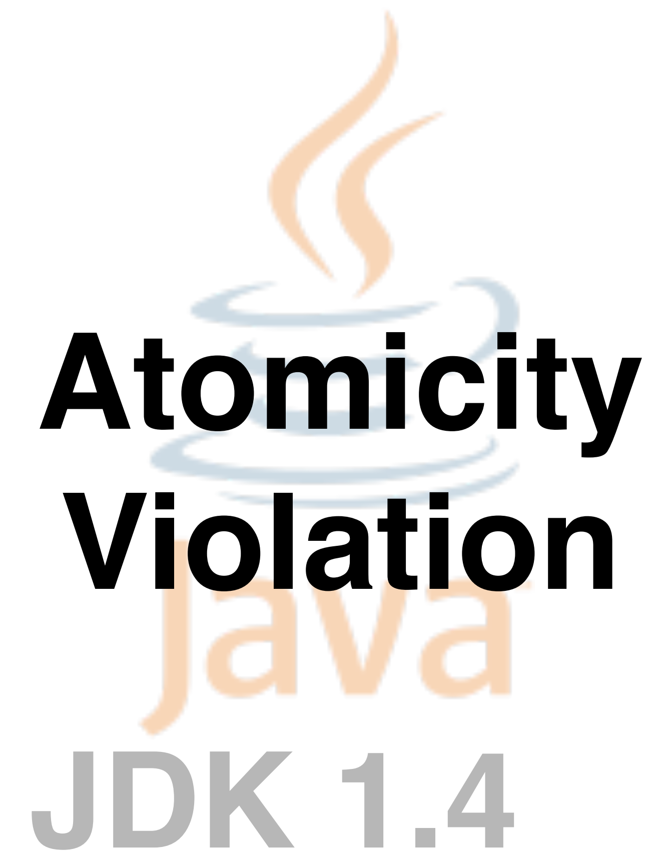
Evaluating LWS and CTraps

JDK 1.4

pbzip2

Transmission

Debugging with LWS



Atomicity Violation

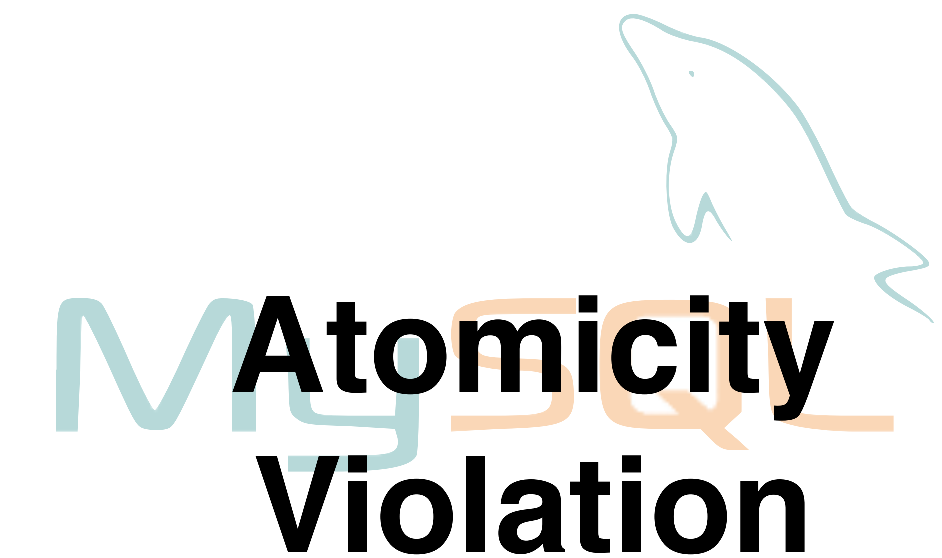
Ordering Error

Ordering Error

Ordering Error

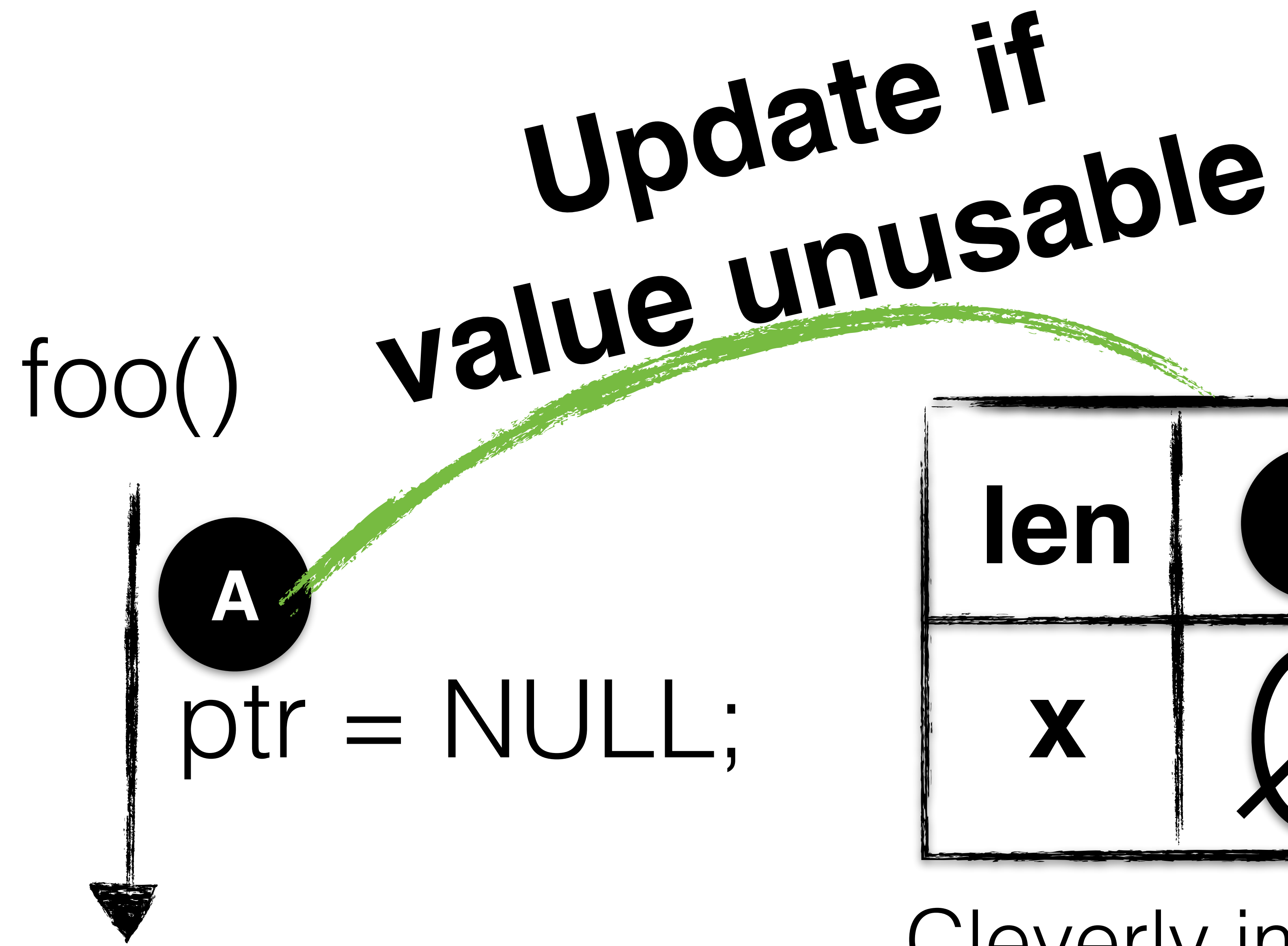
Atomicity Violation

Atomicity Violation

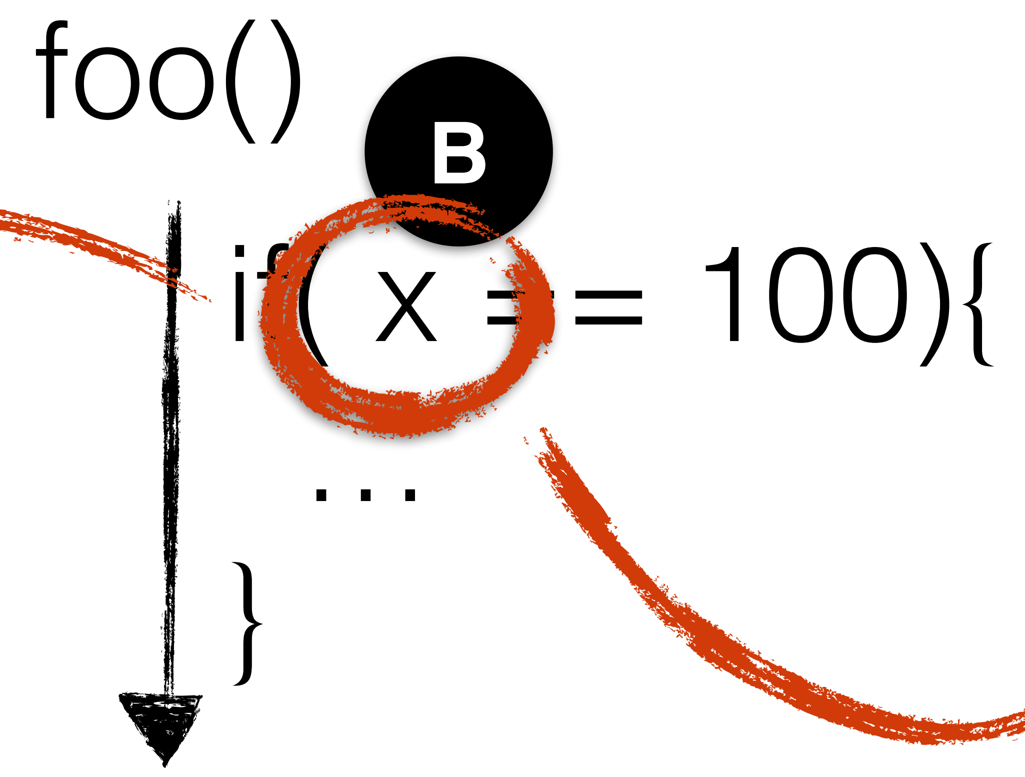


Atomicity Violation

Debugging with LWS



Cleverly implemented
using value 'piggybacking'



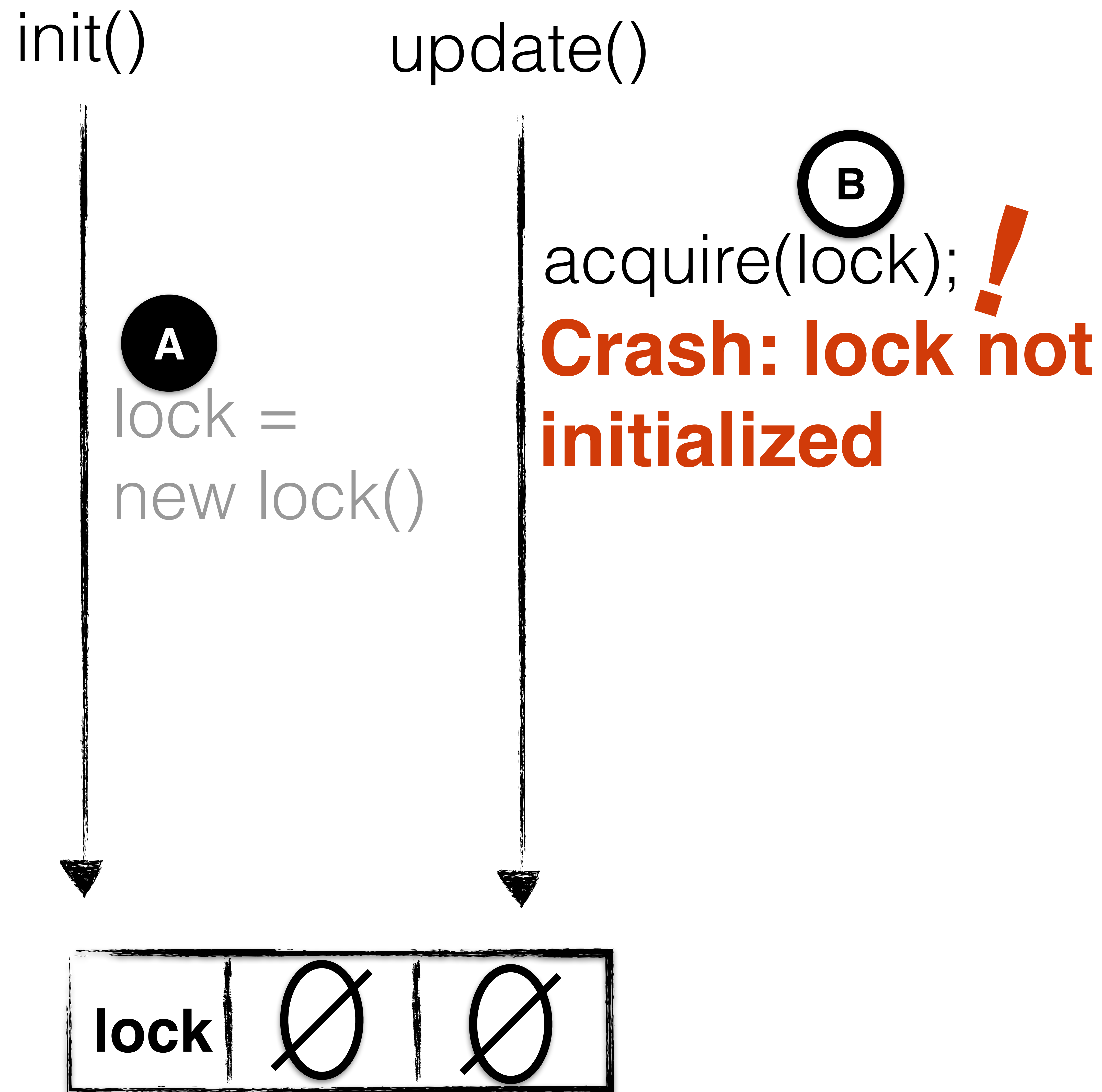
"Undefined value originating at B used in conditional"

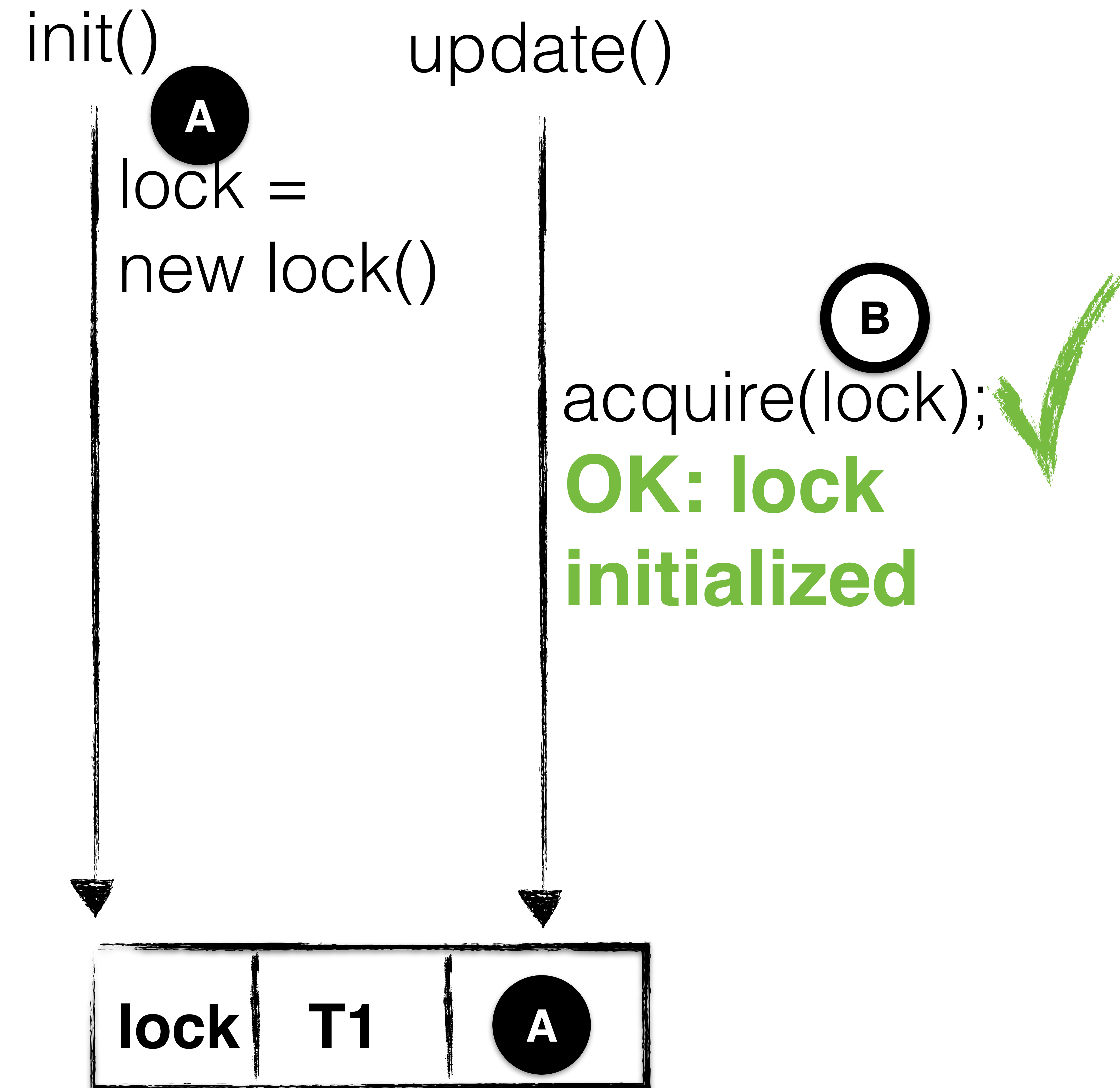
Comparison Point: **Bad Value Origin Tracking**

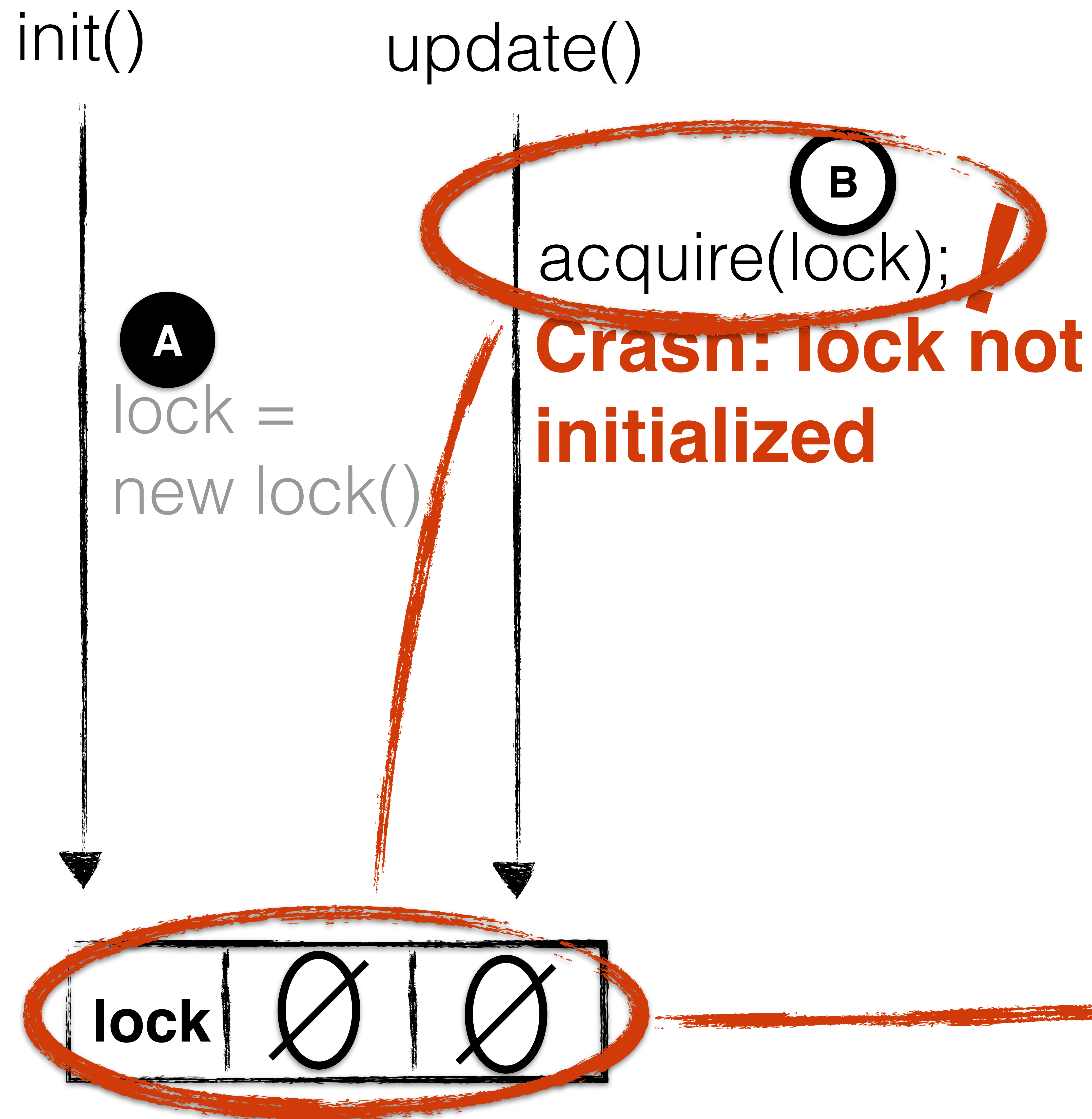
[Bond, et al OOPSLA '07]

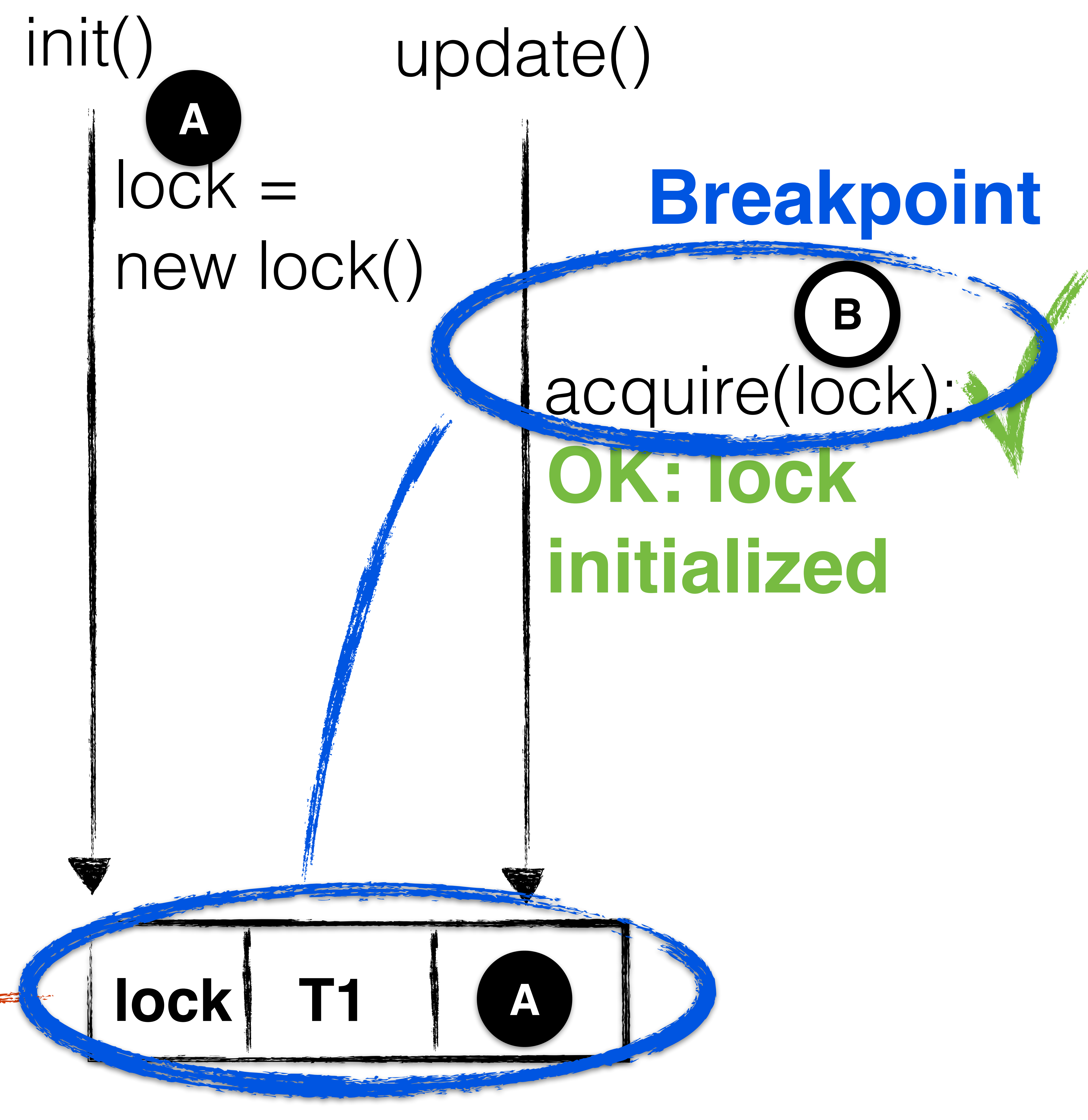
Bug	LWS	BVOT
Atomicity Violation	✓	✗
Ordering Error	✓	✓
Ordering Error	✓	✗
Ordering Error	✓	✗
Atomicity Violation	✓	✗
Atomicity Violation	✓	✓*
Atomicity Violation	✓	✗

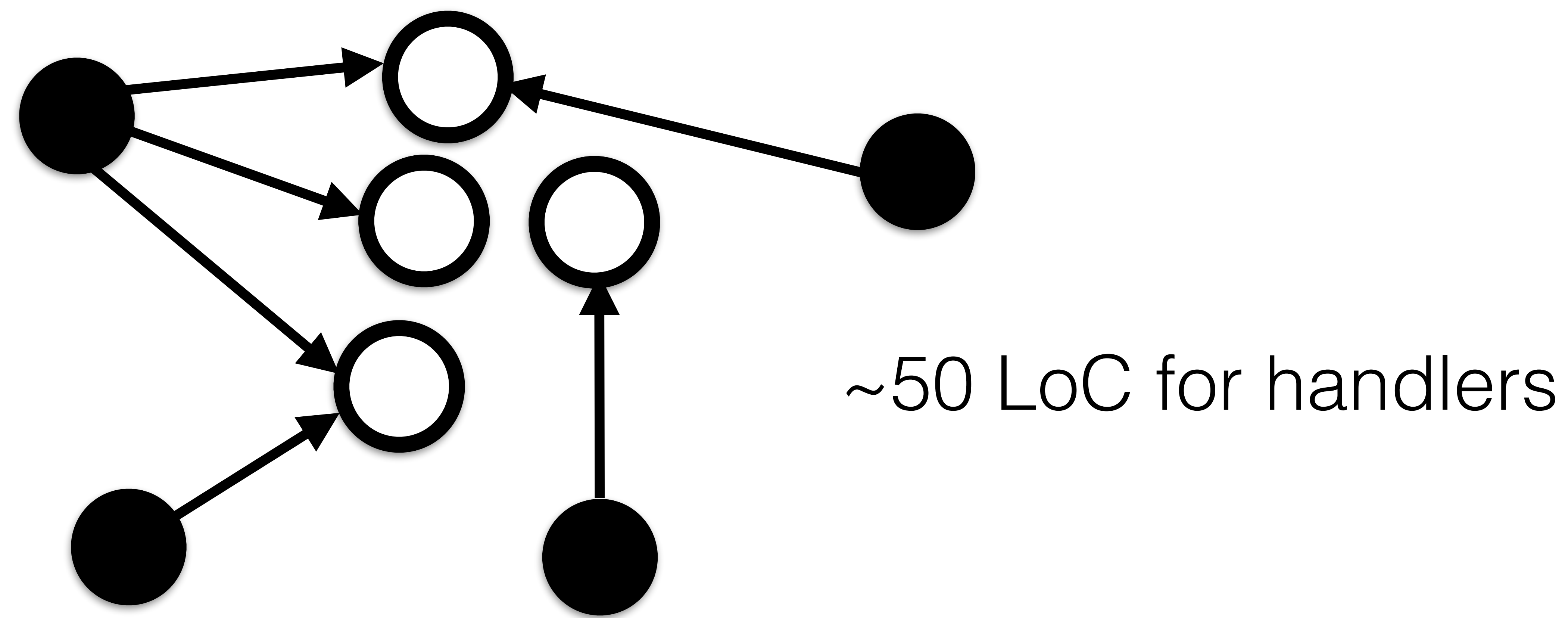
LWS tracks all values, BVOT only unusable ones


Failing Execution
init()
update()
B
acquire(lock);
!
Crash: lock not initialized
A
lock =
new lock()
lock
Ø
Ø

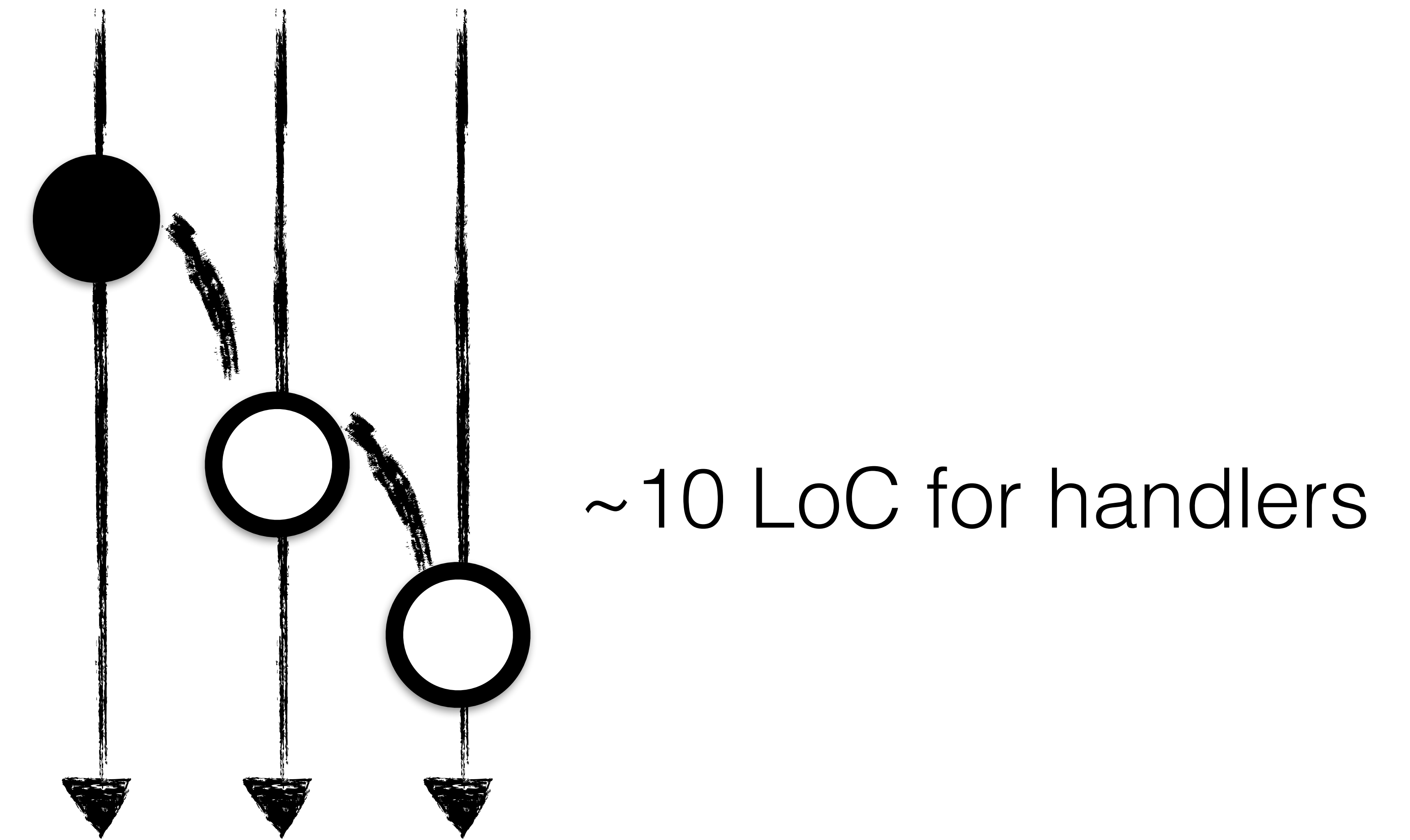

Non-Failing Execution
init()
update()
A
lock =
new lock()
B
acquire(lock);
OK: lock initialized
lock
T1
A


Failing Execution
init()
update()
B
acquire(lock);
Crash: lock not initialized
A
lock =
new lock()
lock


Non-Failing Execution
init()
update()
A
lock =
new lock()
Breakpoint
B
acquire(lock);
OK: lock initialized
lock
T1
A



Communication Graph Collection
[Lucia, MICRO ’09; PLDI ’11;
Shi, OOPSLA ’10; Gao, SC ’07;]



CCI-Prev
[Jin, OOPSLA ’10]

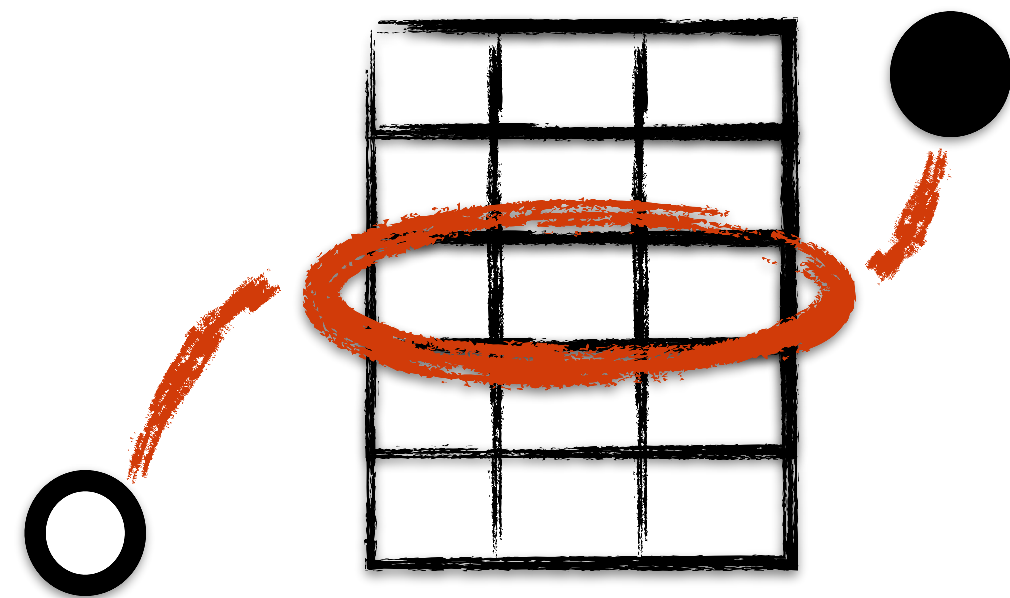
CTraps Supports Useful Analyses

LWS has overhead low enough for production use
slowdown
4
3.5
3
2.5
2
1.5
1
0.5
0
50% Overhead
10% Overhead
9%
49%
MySQL
Apache
memcached
LevelDB
AMean
GMean
blackscholes
dedup
canneal
streamcluster
x264
fluidanimate
ferret
vips
swaptions
AMean
GMean

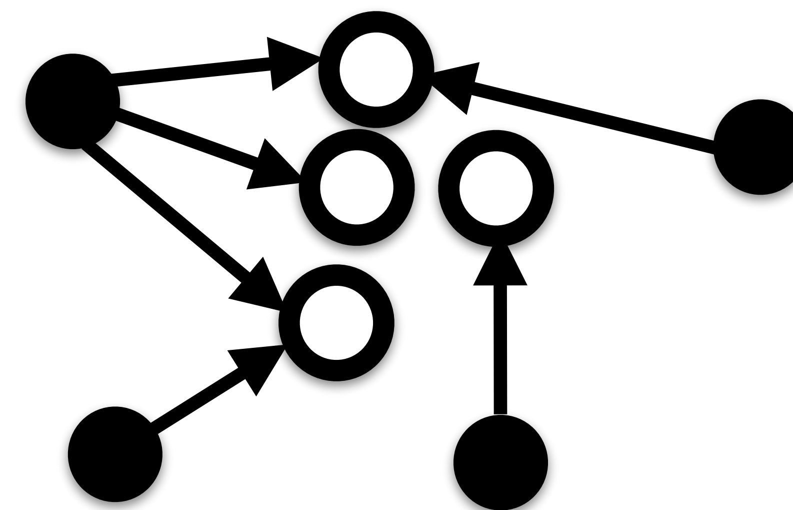
CTraps has practical overhead that scales with analysis complexity

Systems should track data provenance information

LWS helps with
Debugging



CTraps enables useful
Analysis

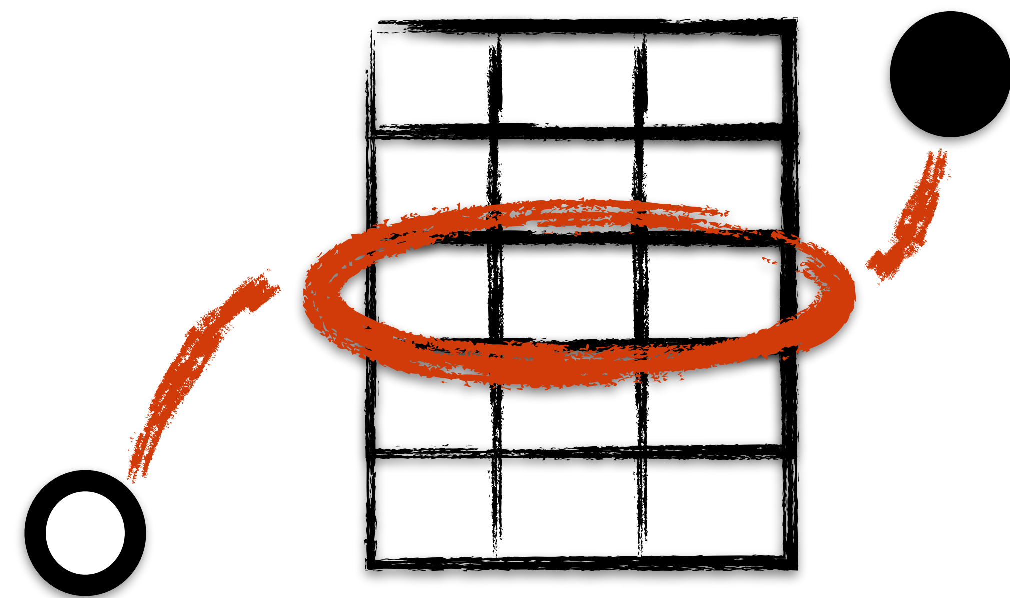


LWS & CTraps have
Efficiency
sufficient for production

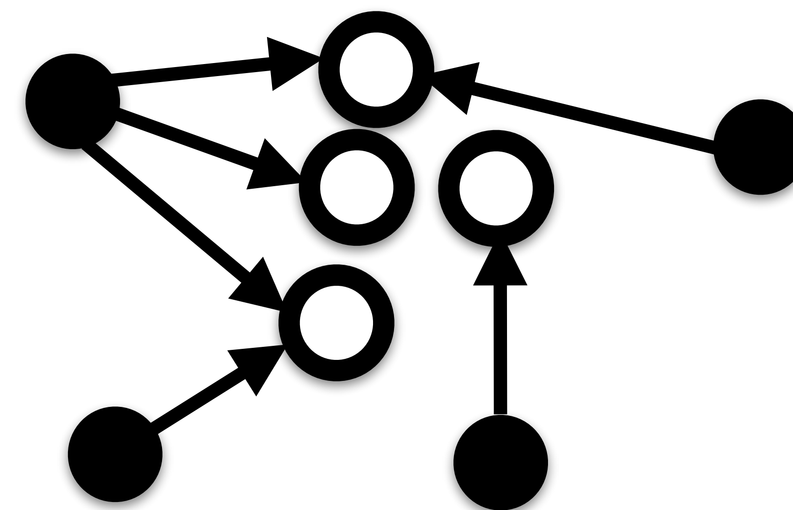
https://github.com/blucia0a/CTraps-gcc

https://gcc.gnu.org/wiki/plugins

LWS helps with
Debugging



CTraps enables useful
Analysis



LWS & CTraps have
Efficiency
sufficient for production

Data Provenance Tracking for Concurrent Programs

Brandon Lucia | Carnegie Mellon University, Dept. of ECE

work done in cooperation with ***Luis Ceze @ University of Washington, Dept. of CSE***