6.S081 : Locks

app wants to multiple cores

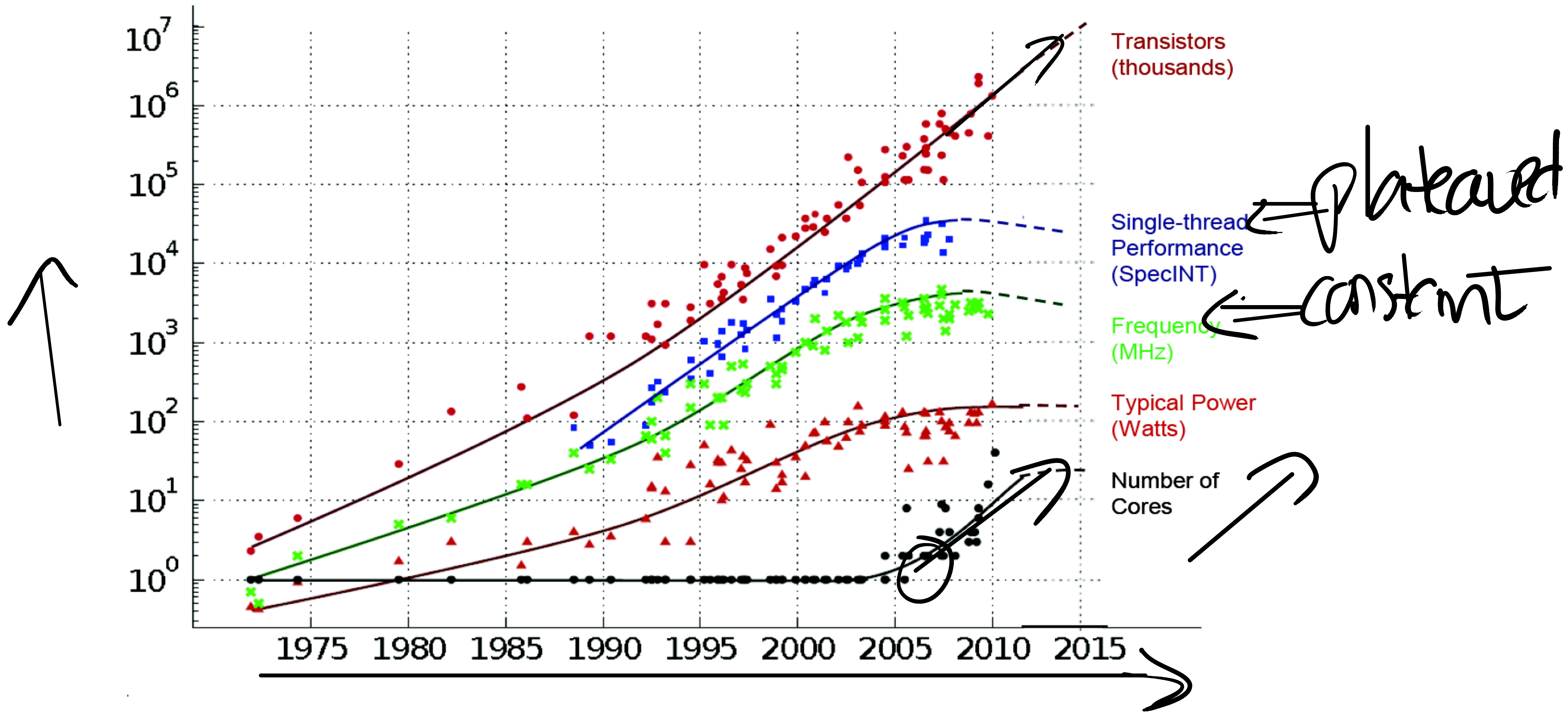
kernel must handle parallel systemcalls

access shared data structures in parallel

$\Rightarrow$ Locks for correct sharing

Locks can limit performance

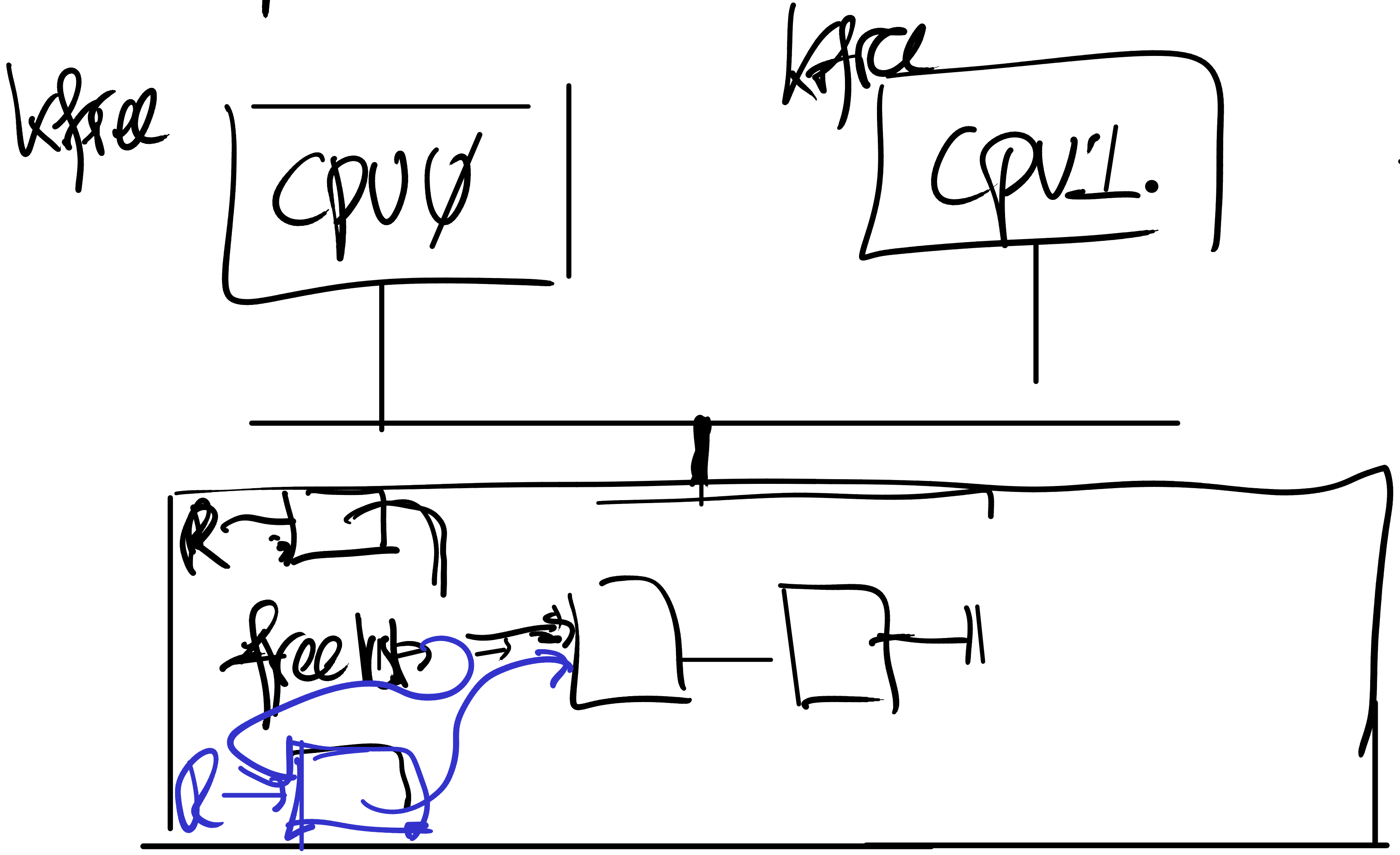
35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Why locks?

Avoid race conditions



lost page

Lock abstraction

acquire(&l) → $\boxed{\text{one process can acquire lock.}}$

struct lock {

}

acquire(&l)
↕ critical section
release(&l)

Programs have many locks ⇒ more parallelism

When to lock?

Conservative rule: 2 processes
access a shared data structure +
one is writer ⟹ lock data structure

<u>too strict:</u> lock-free programming

<u>too loose:</u> printf("_ _")

Could locking be atomic.

every struct has a lock → too rigid

Rename("d1/x", "~~d1~~d2/y")

1 [lock ~~d1~~ d1 ; erase x ; release d1 ← file doesn't exist

2 [lock ~~d2~~ d2 ; add y ; release d2

Need: lock d1+d2 ; erase+add, release ~~d1~~ d1+2

Lock perspectives

1) locks $^{\text{help}}$ avoid lost updates
2) locks make multi-step op atomic
3) locks help maintain invariant

Deadlock

acquire(ℓ)
⋮
acquire(ℓ) ←
↓
deadlock

d1, d2   d1, d2

CPU1	CPU2
rename("d1/x", "d2/y")	rename("d2/a", "d1/b")
acquire(d1)	acquire(d2)
⋮	⋮
acquire(d2)	acquire(d1)

deadlock

Solutions: order locks
acquire locks in order

Locks vs. modularity

Lock ordering $\Rightarrow$ global

$$\begin{tikzcd} M_1.g() \arrow[r, dash, bend left] & M_2.f() \end{tikzcd}$$

locks M_2 uses

Internals of M_2 in terms of locks must be visible to M_1

Locks vs. performance

① Start with coarse-grained locks

② Measure

↳ lock is contended

↳ redesign

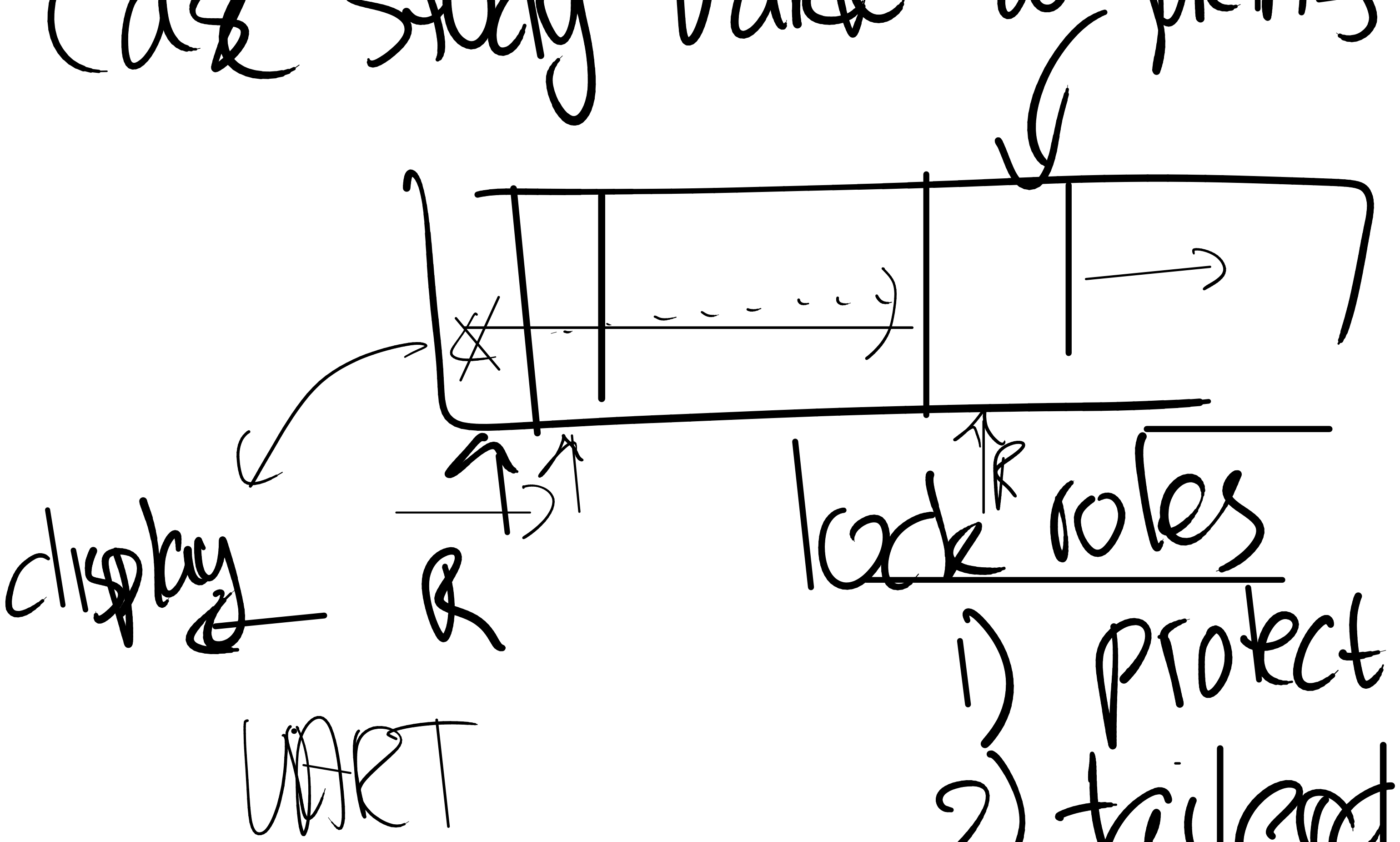
Need to split up data structures

Best split is a challenge

May need to rewrite to code too

⇒ lot of work!

Case study uart w printf

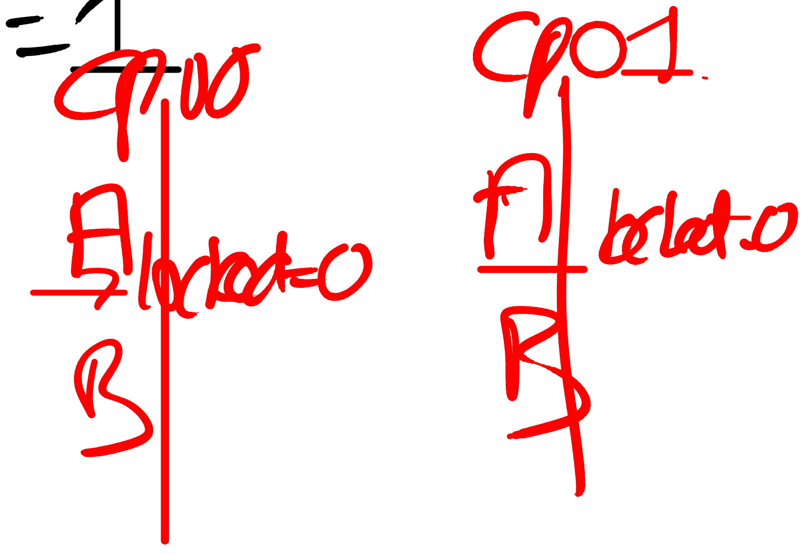


lock roles

1) protect this datastr
2) tailend is in flight
3) hw registers have one writer

Broken acquire(struct lock *l) {

```
      while (1) {
A         if(l->locked==0){   <———— race.
B             l->locked=1;
              return;
          }
      }
}
```



Hardware test-and-set support:

atomswap addr, r1, r2

```
lock addr
    tmp ← *addr
    *addr ← r1
    r2 ← tmp
unlock
```

impl[illegible]
dependant
on
mem system

Memory ordering

acq locked ← 1

X ← X + 1

relax locked ← 0 ?

Single serial execution ok

wrong in concurrent execution

Wrap up :

- locks good for correctness
  can be bad for perf
- locks complicate programming
- don't share if you don't have to
- start with coarse-grained
- use race detector

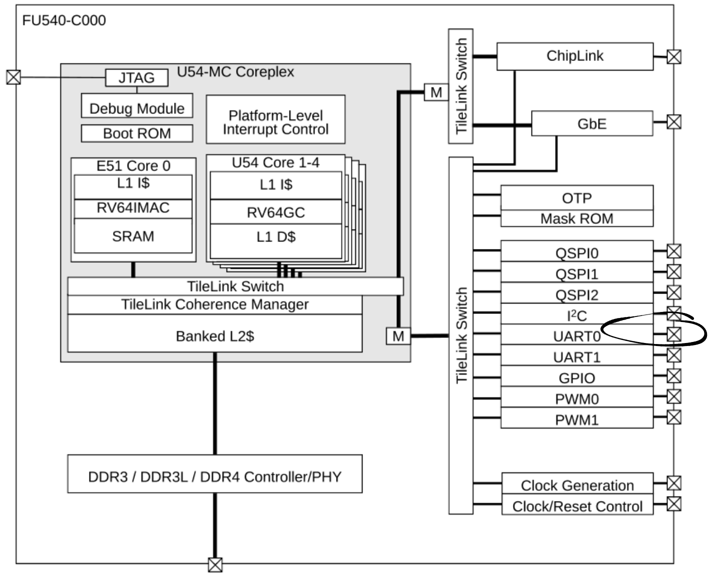


Figure 1: FU540-C000 top-level block diagram.