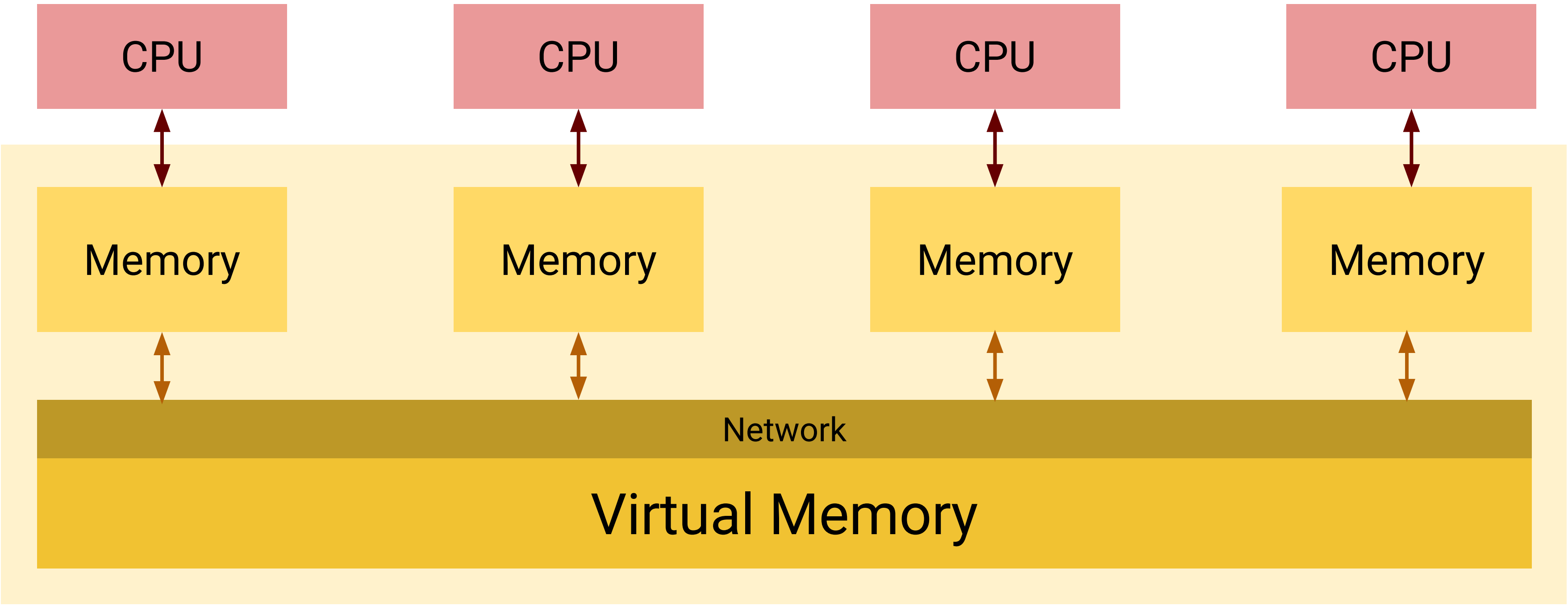
Distributed Shared Memory (DSM)

Robert Gasparyan, Angela Gong, Judson Wilson

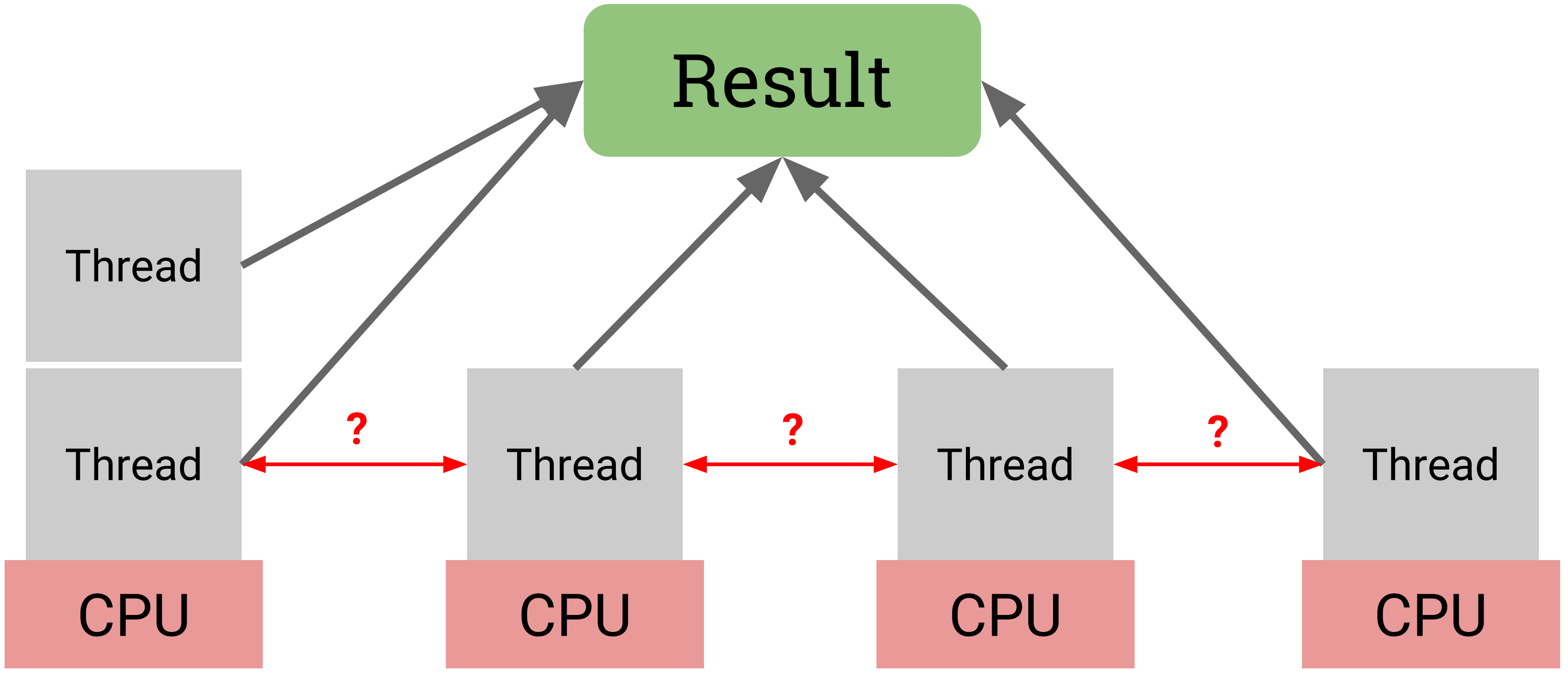
CS 240, Spring 2015

What is DSM?

- **Physically separate** memory addressed as **one shared address space**
- Memory shared on page-by-page basis

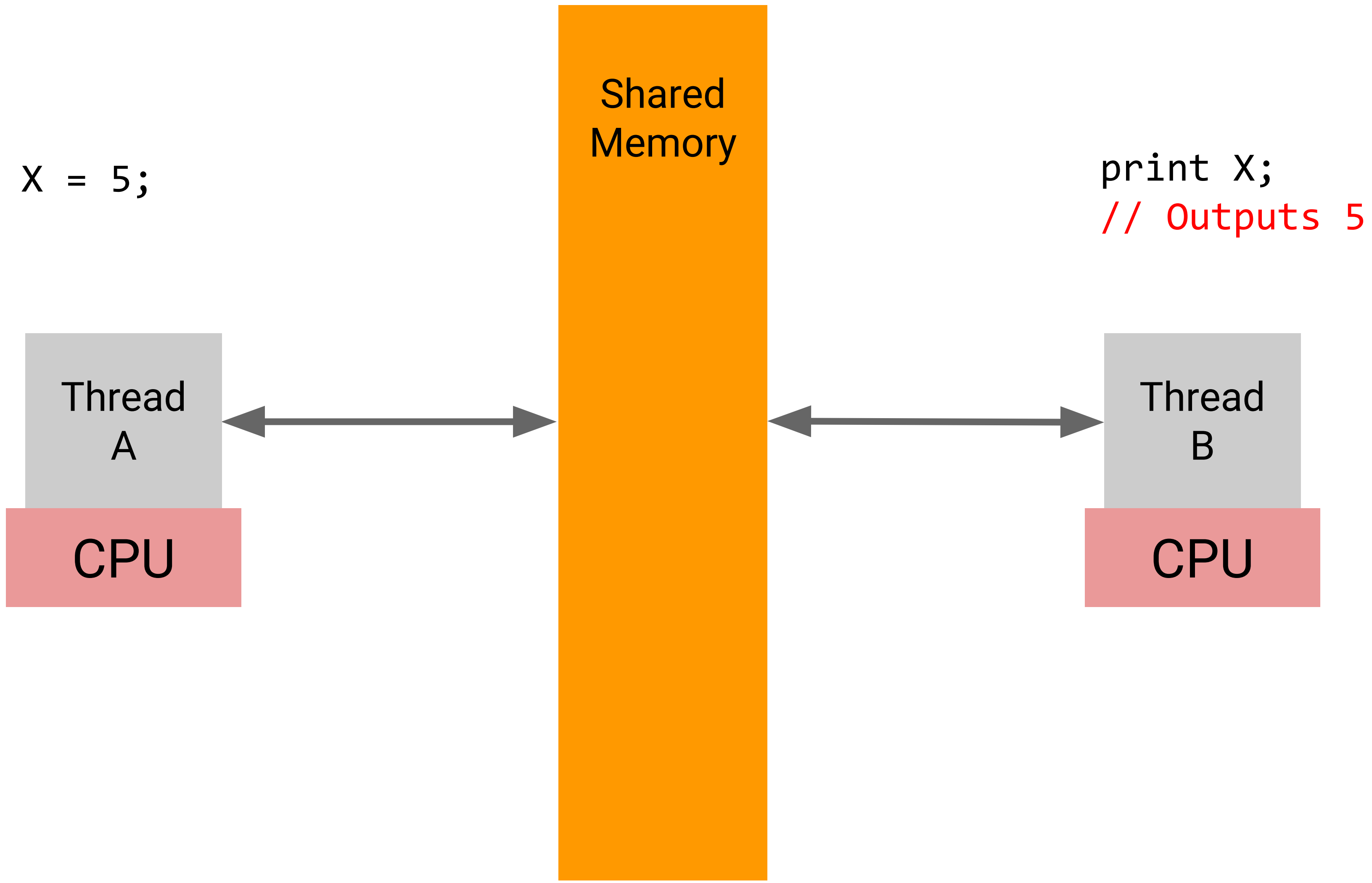


Problem

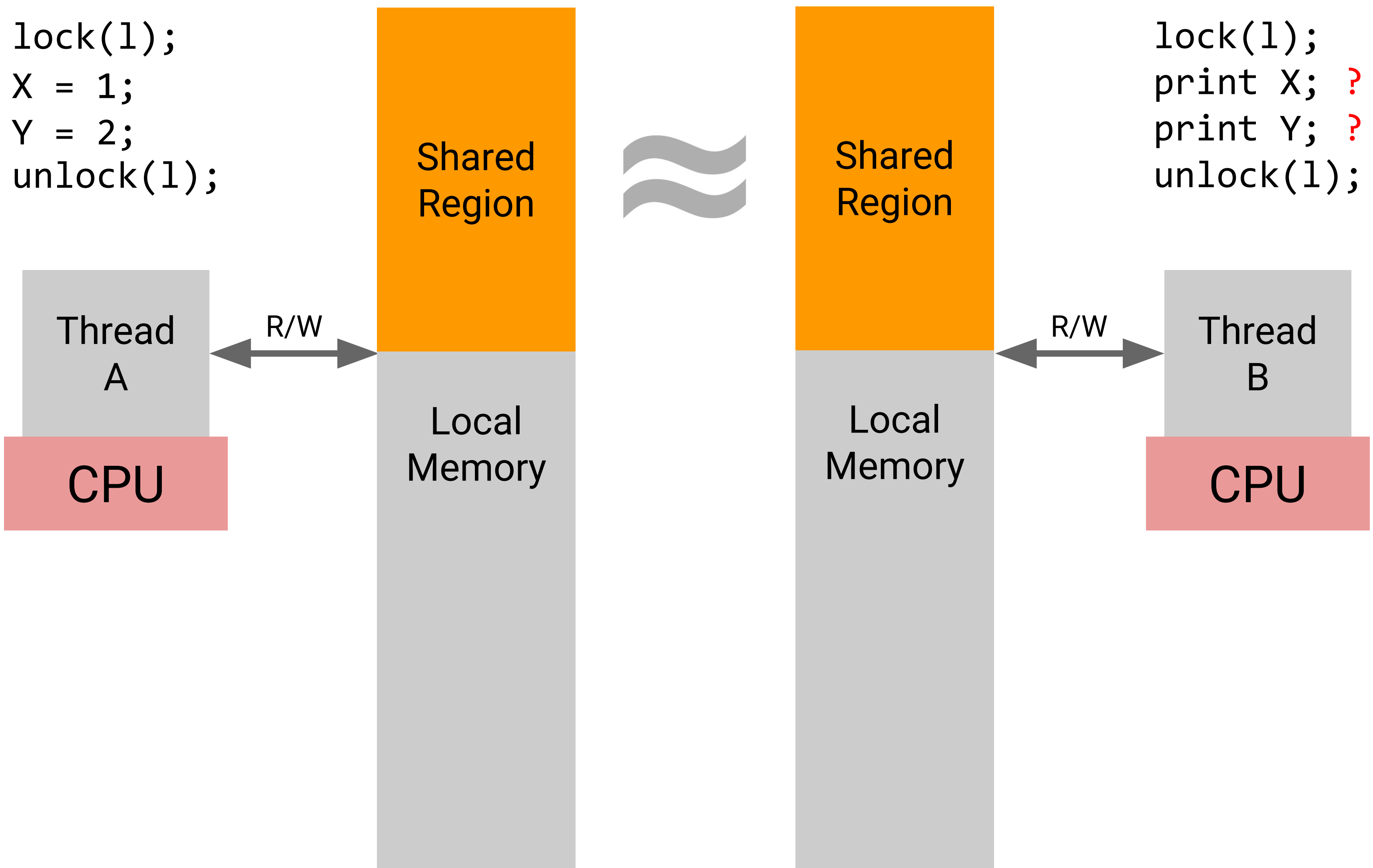


- Make use of multiple machines
- Manage dependencies

DSM: Simple Interface



Consistency Model

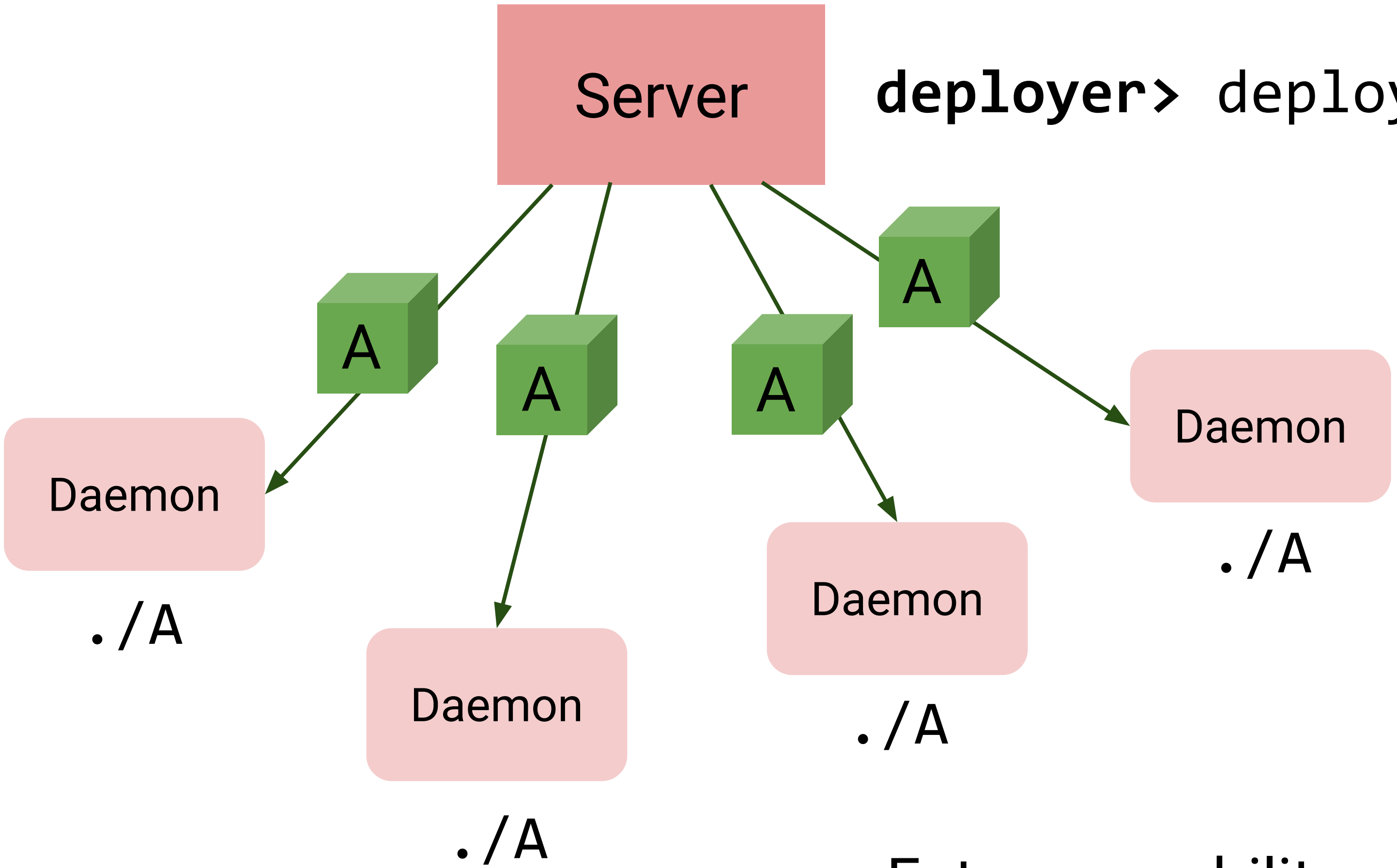


Release Consistency

- Critical sections protected by same lock execute sequentially
- All changes from previously protected regions guaranteed to be visible
- Saves network traffic because don't need to synchronize until lock is released

Design and Implementation

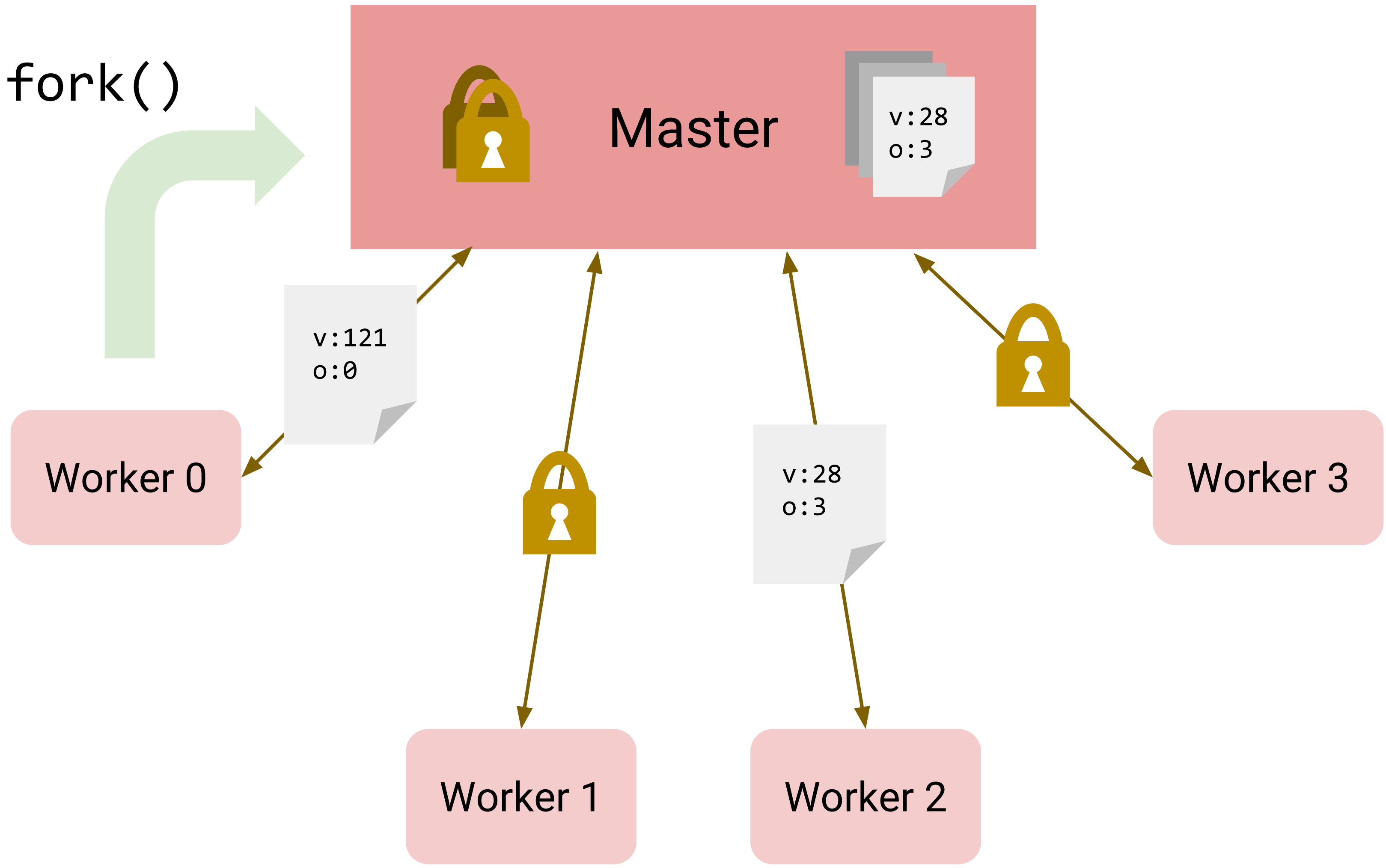
Deployment of Binaries



deployer> deploy A

Future capability:
deployer> deploy gdb A

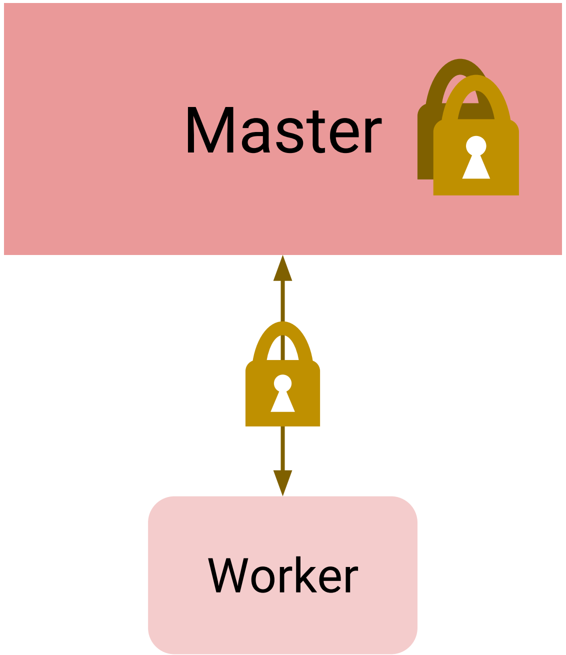
Master: Locks and Page Info



Locks

Lock Requests

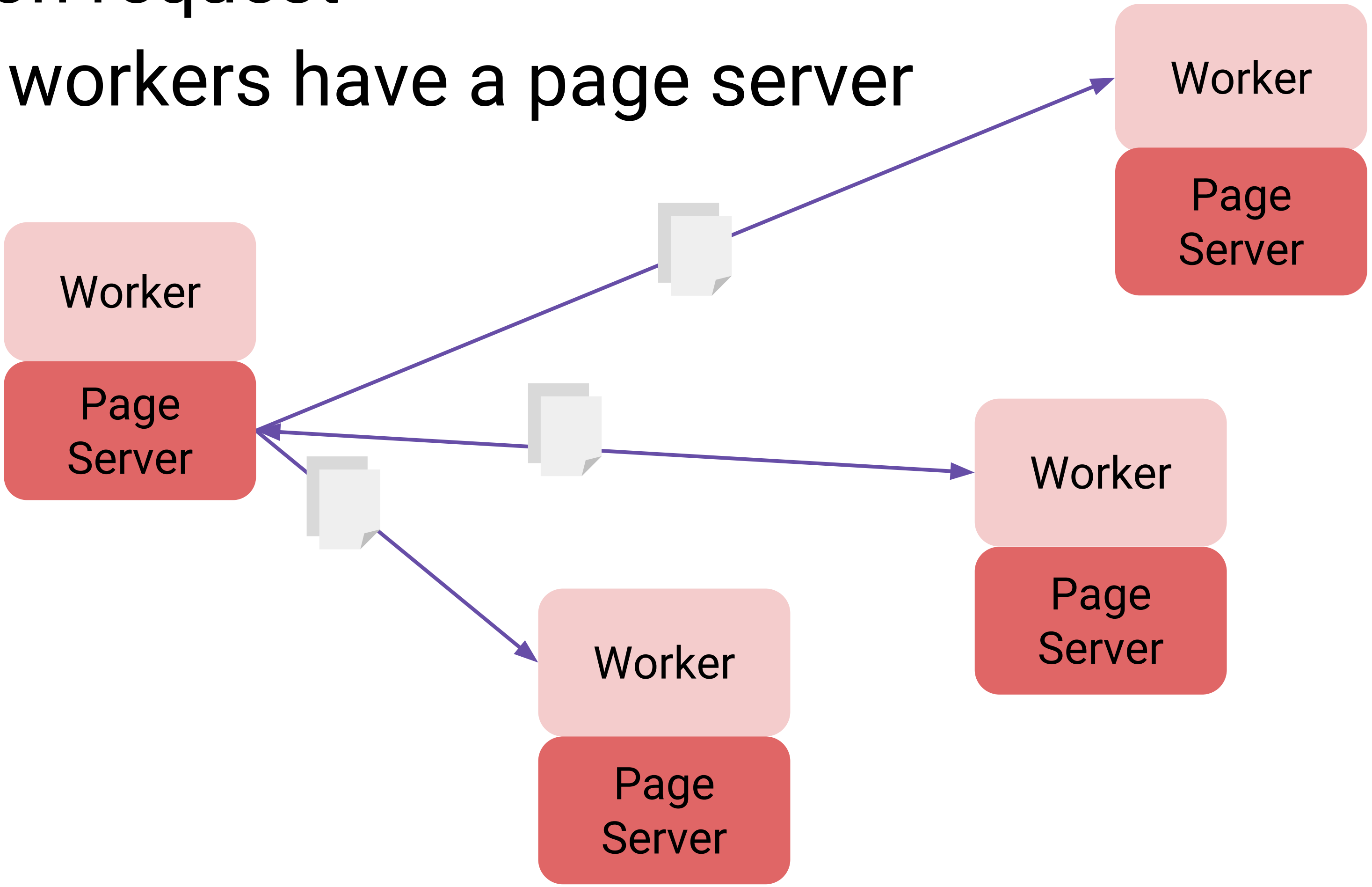
- Spawn thread
- Wait on `pthread_mutex_lock()`
- Reply after acquired



Similar for unlock

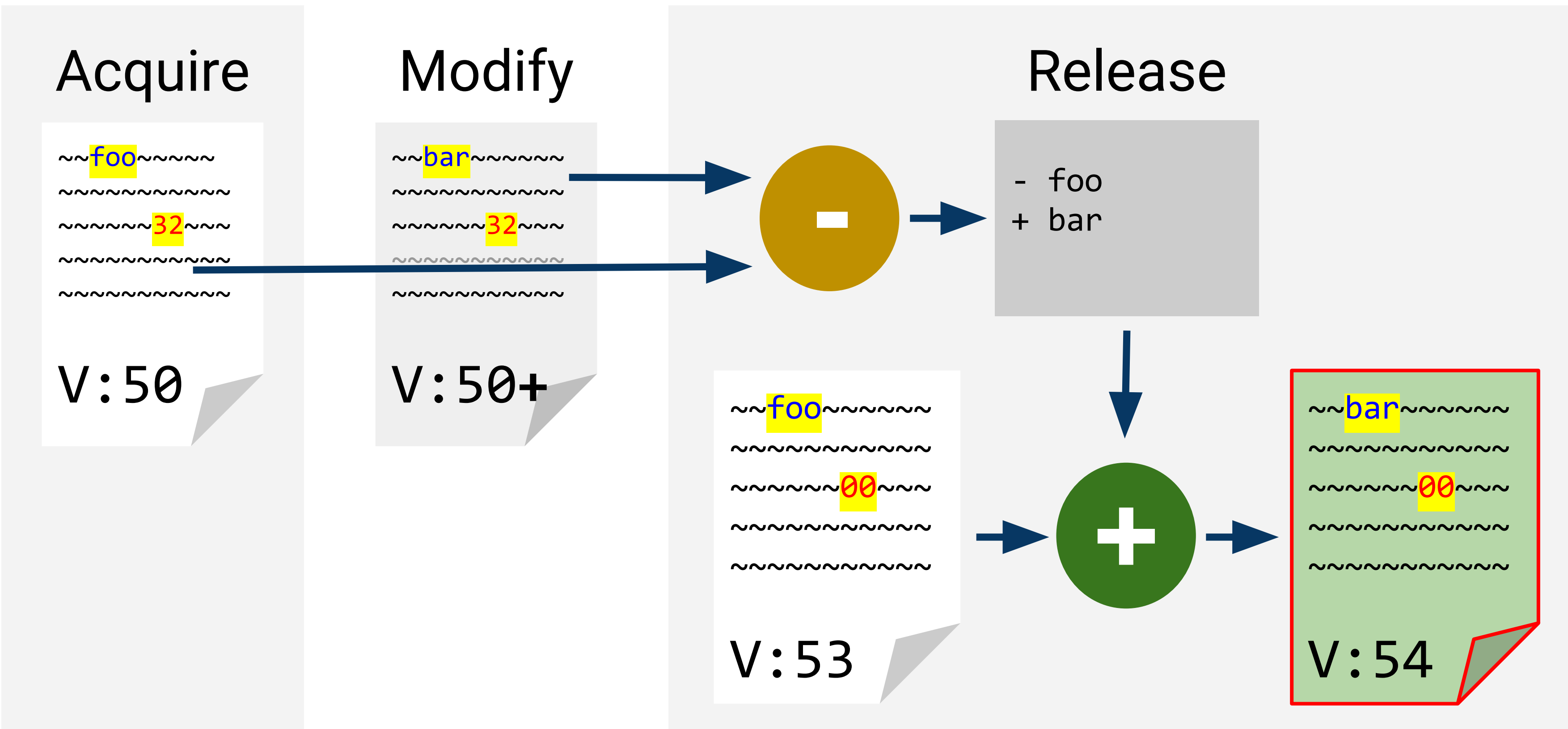
Page Servers: Page Contents

- Background thread that distributes page data upon request
- All workers have a page server



Patches

Implicitly locked memory can be smaller than a page. Used diff-patches to merge modifications



Transparent Interface

Fault handlers, lock/unlock:

- `lock()` starts (lazy) `Acquire`
  - `mprotect` to protect pages
- `SIGSEGV` fault to catch first read/write
  - 1st: Upgrade to `READ` access → get latest version
  - 2nd: Upgrade to `WRITE` → mark modified
- `unlock()` does `Release`
  - Pull versions, merge modified pages, create new version

Transparent Shared Memory

Across Processes

```
#include <pthread.h>
...
// Shared region setup
void *r;
size_t len = 4 * 1024;
r = mmap(NULL, len, ...,
   MAP_SHARED|MAP_ANONYMOUS, ...);
int pid = fork();


...
// Do stuff
pthread_mutex_lock(&lock);
do_work(pid);
pthread_mutex_unlock(&lock);
...

if (pid) wait(pid);
```

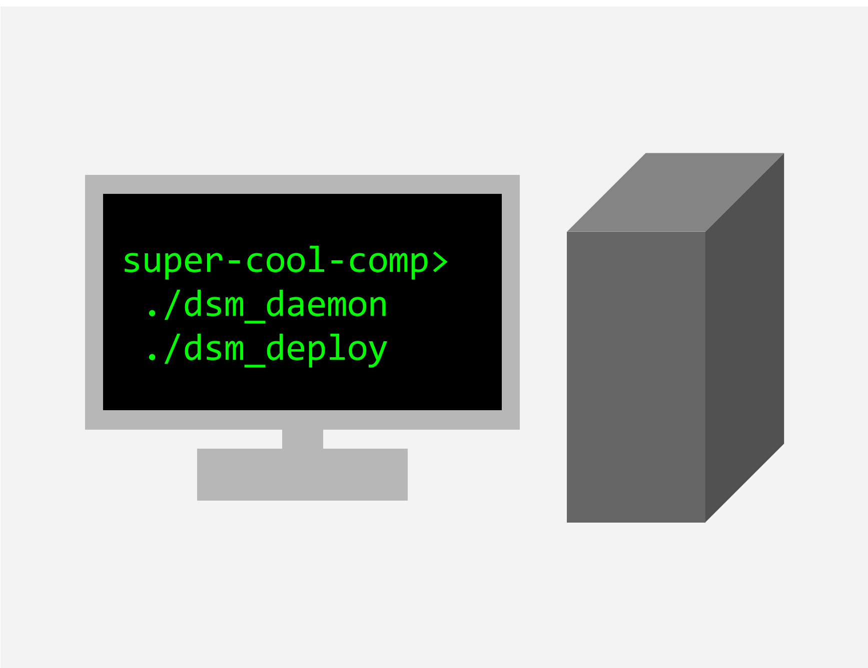
Across Network (DSM)

```
#include <dsm.h>
...
// Shared region setup
void *r = (void *)0x400000000000;
size_t len = 4 * 1024;
dsm_share(r, len);
dsm_start(argc, argv);



...
// Do stuff
dsm_lock(LOCK);
do_work(machine_number);
dsm_unlock(LOCK);
...

dsm_wait();
```

Benchmarks

Benchmark Setup

- Compare performance of single machine versus DSM.



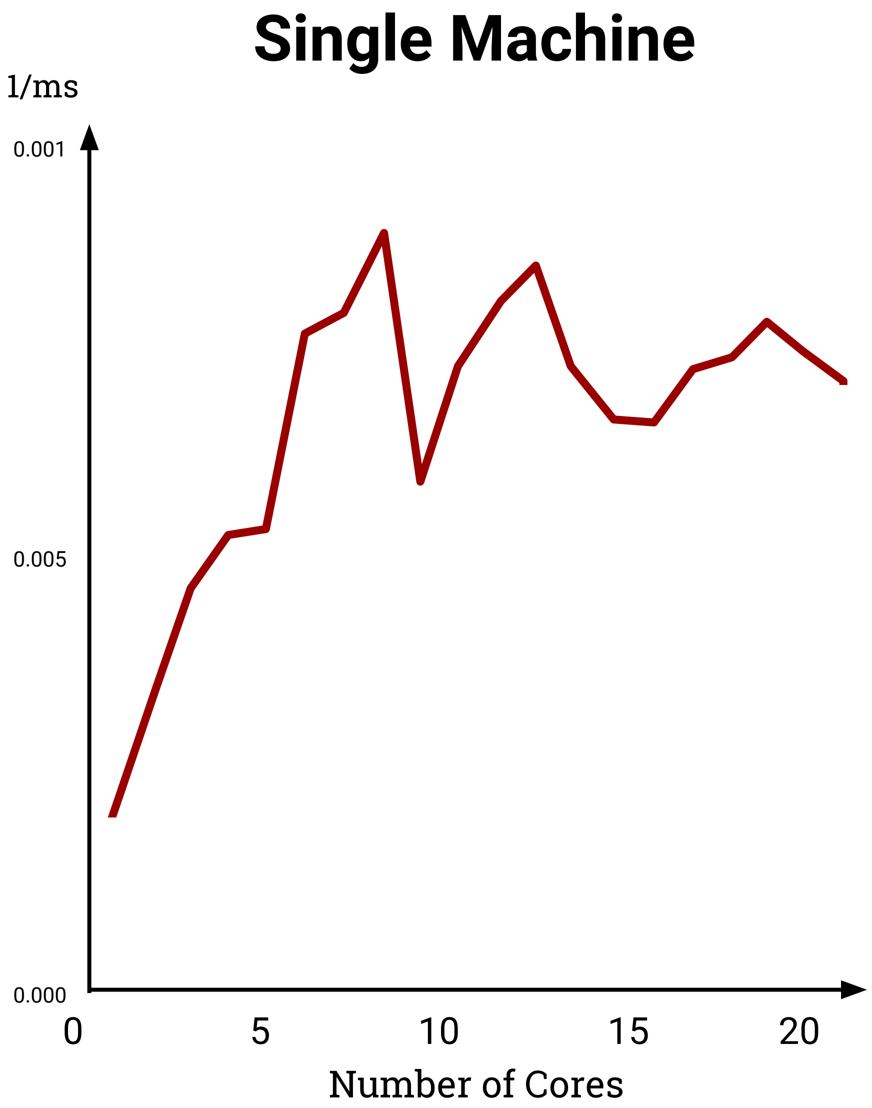
Single Machine

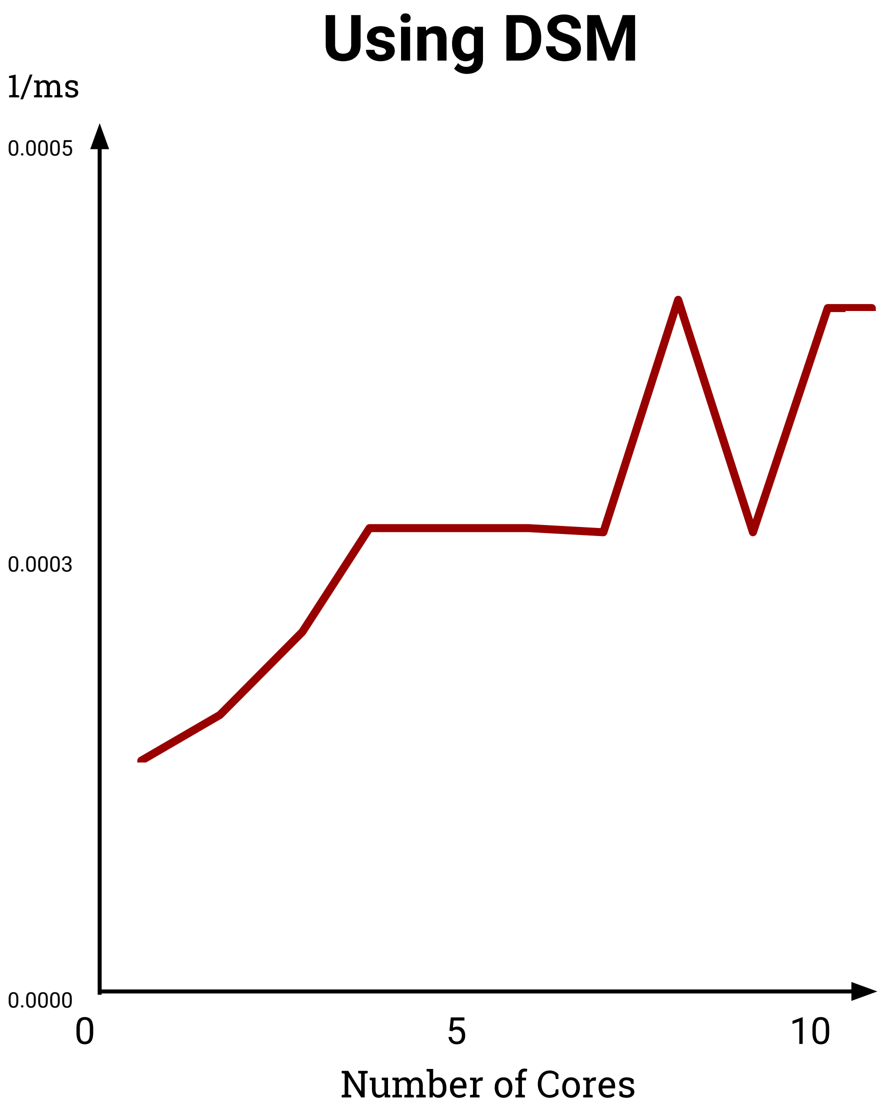
Using DSM

Matrix Multiplication

$$\begin{pmatrix} -9 & 7 & -1 \\ 6 & -5 & 2 \\ 6 & -4 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 3 \\ 2 & -1 & 4 \\ 2 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Matrix Multiplication





Word Count

▶ **32.** [*M32*] How many of the genlex listings of (s,t)-combination strings $a_{n-1}\ldots a_1a_0$ (a) have the revolving-door property? (b) are homogeneous?

33. [*HM33*] How many of the genlex listings in exercise 31(b) are near-perfect?

34. [*M32*] Continuing exercise 33, explain how to find such schemes that are as near as possible to perfection, in the sense that the number of "imperfect" transitions $c_j \leftarrow c_i \pm 2$ is minimized, when s and t are not too large.

84. [*HM27*] If $T = \binom{2t-1}{t}$, prove the asymptotic formula

$$\kappa_t N - N = \frac{T}{t}\left(\tau\left(\frac{N}{T}\right) + O\left(\frac{(\log t)^3}{t}\right)\right) \qquad \text{for } 0 \le N \le T.$$

85. [*HM21*] Relate the functions $\lambda_t N$ and $\mu_t N$ to the Takagi function $\tau(x)$.

86. [*M20*] Prove the law of spread/core duality, $X^{\sim +} = X^{\circ \sim}$.

42. A. Vershik [*Functional Anal. Applic.* **30** (1996), 90–105, Theorem 4.7] has stated the formula

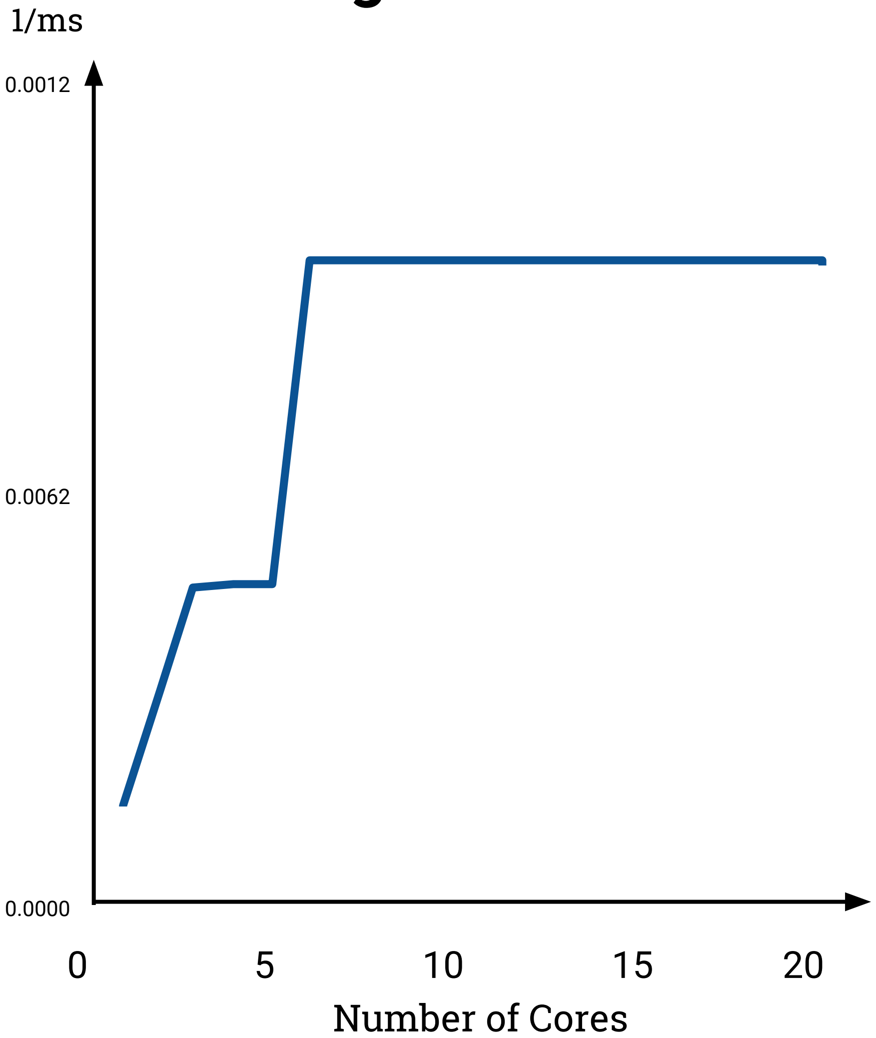
$$\frac{1-e^{-c\varphi}}{1-e^{-c(\theta+\varphi)}}e^{-ck/\sqrt{n}} + \frac{1-e^{-c\theta}}{1-e^{-c(\theta+\varphi)}}e^{-ca_k/\sqrt{n}} \approx 1,$$

where the constant c must be chosen as a function of θ and φ so that the area of the shape is n. This constant c is negative if $\theta\varphi < 2$, positive if $\theta\varphi > 2$; the shape reduces

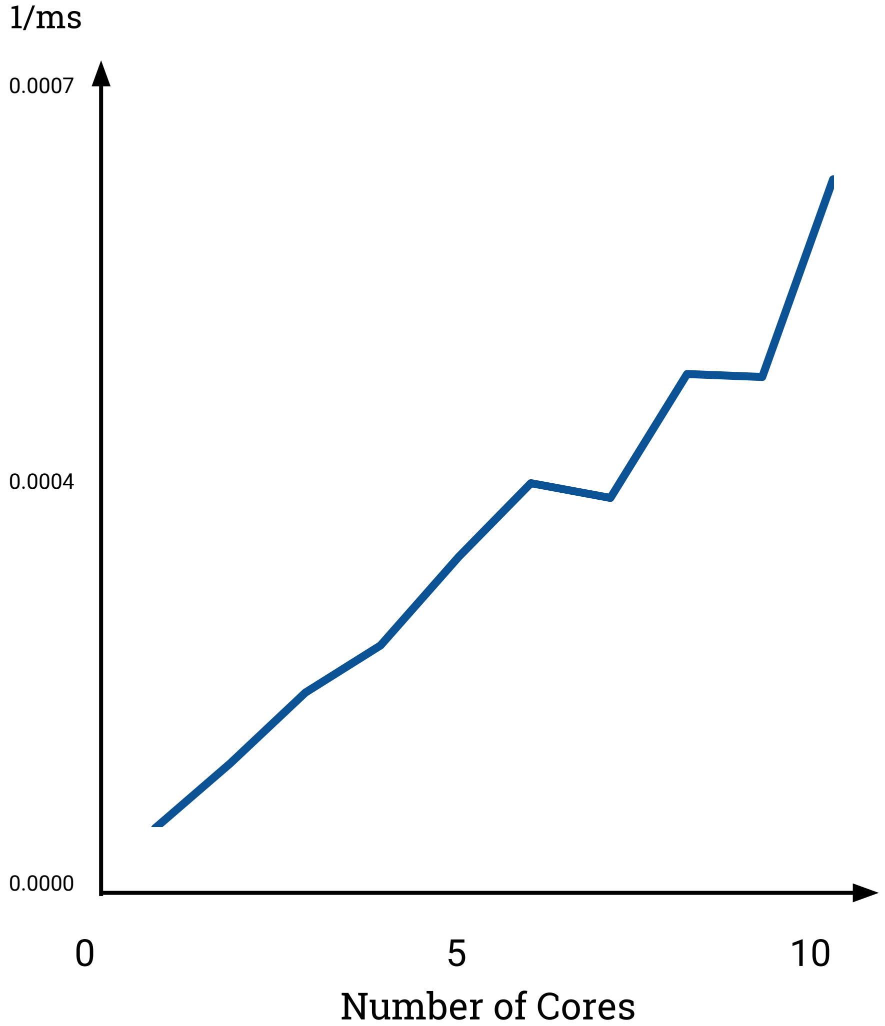
Donald Ervin Knuth: The Art of Computer Programming: Generating all Combinations and Partitions

Word Count

Single Machine



Using DSM



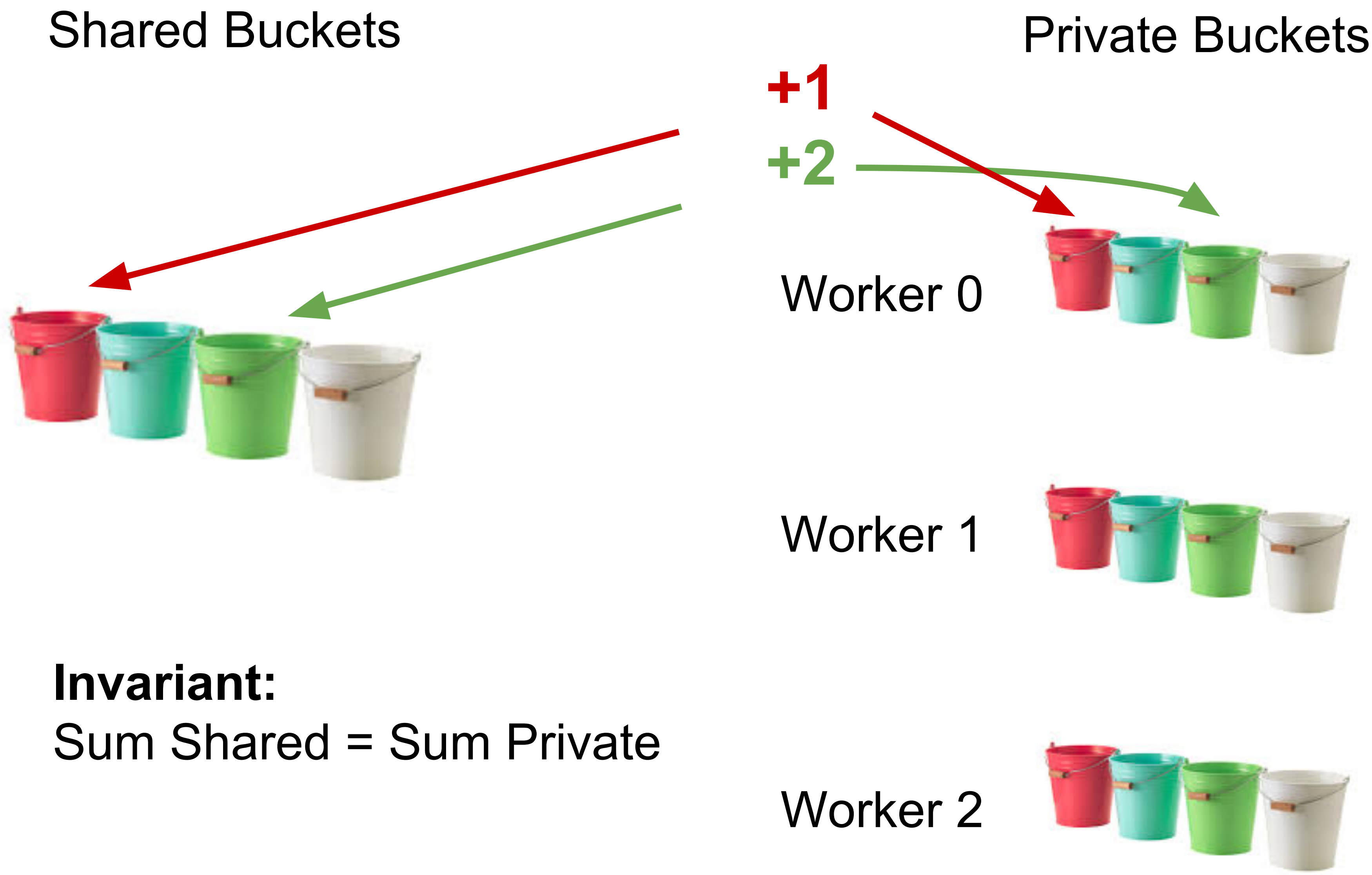
Demo!

Conclusion

- We made transparent DSM!
- Focus: Correctness first, then scalability
- Tedious:
  - C data structures
  - Message passing / handling


Questions?

Bonus: Correctness Test



Invariant:

Sum Shared = Sum Private

Bonus: Correctness Test

Nested Increment

```
dsm_lock(LA);
dsm_lock(LC);
++*C; ++c_priv;
dsm_unlock(LC);
++*A; ++a_priv;
dsm_unlock(LA);
```

Serial Transfer 1 Count

```
dsm_lock(LA);
++*A;
dsm_unlock(LA);
dsm_lock(LB);
--*B;
dsm_unlock(LB);
```

Nested Transfer All Counts

```
dsm_lock(LA);
temp = *A;
*A = 0;
dsm_lock(LD);
*D+=temp;
dsm_unlock(LA);
dsm_unlock(LD);
```