Cache-Adaptive Analysis

Michael A. Bender[1] Erik Demaine[4] Roozbeh Ebrahimi[1]
Jeremy T. Fineman[3] **Rob Johnson[1]**
Andrea Lincoln[4] Jayson Lynch[4] Samuel McCauley[1]

1 Stony Brook University

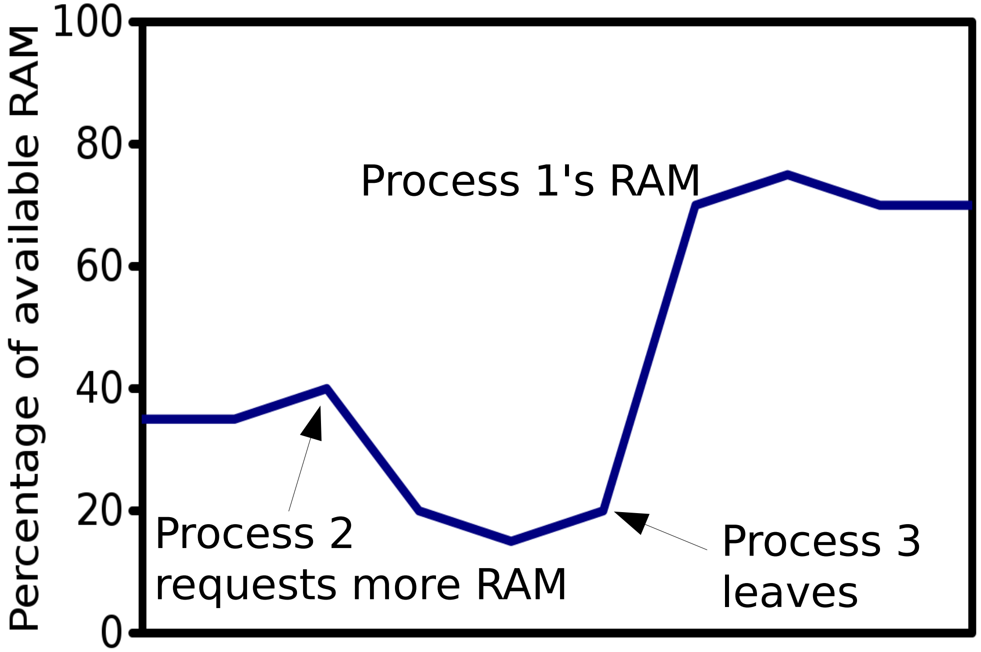
3

4

Available Memory Can Fluctuate in Real Systems

Memory fluctuations are common

- Jobs starting and stopping
- Irregular parallel programs
- Any time-sharing system



Performance can be lost when algorithms can't adapt to changes in available memory

- Thrashing (when available memory shrinks)
- Underutilization (when available memory grows)

Adapting to Memory Changes: Empirical

Database papers on adaptive sorting or joins:

- Empirical good, but not provably good.
- Rarely present in production systems, despite the need.

[Pang, Carey, Livny, VLDB 93], [Zeller+Gray VLDB 90], [Zhang+Larson VLDB 97], [Zhang+Larson, CASCON 96], [Pang, Carey, Livny, SIGMOD/COMAD 93], [Graefe 13]

Adapting to Memory Changes: Theoretical

Barve and Vitter **[98, FOCS 99]** generalize the I/O model **[Aggarwal+Vitter ’88]** to allow RAM to change size.

- These are hard and technically sophisticated results (sorting, FFT, matrix multiplication, etc).
- There’s been little followup work over the last 15 years.

It's hard to write memory-adaptive code and harder to prove bounds about it.

Key Insight

We can design **cache-adaptive** algorithms using **cache-oblivious** algorithms.

Results

Tools for cache-adaptive analysis.

- Extension to external-memory and cache-oblivious models.
- Square profiles and inductive charging
- Worst-case profile analysis
- Machinery for porting progress bounds from DAM to CA model

Characterization theorem for when CO algorithm is CA

- Covers many Akra-Bazzi-style divide-and-conquer algorithms, e.g.
- Matrix multiplication (two versions, one is CA, one is not)
- Matrix transpose
- Jacobi multi-pass filter
- All-pairs shortest paths
- Edit distance
- Longest common substring

Typical Master-theorem-style CO algorithms are either optimal or $\log N$ off.

Cache-oblivious FFT is not CA, but is at most $\log \log N$ off.

Additional Results

Proof that Lazy Funnel Sort [Brodal, Fagerberg 02] is cache adaptive.

Paging results when the cache changes sizes.

- Farthest-in-future is still optimal (cf. [Belady 66]).
- LRU with 4-memory and 4-speed augmentation is competitive with OPT.
- LRU is constant-competitive even if cache hits are not free.
  - And even if OPT gets to perform prefetching.

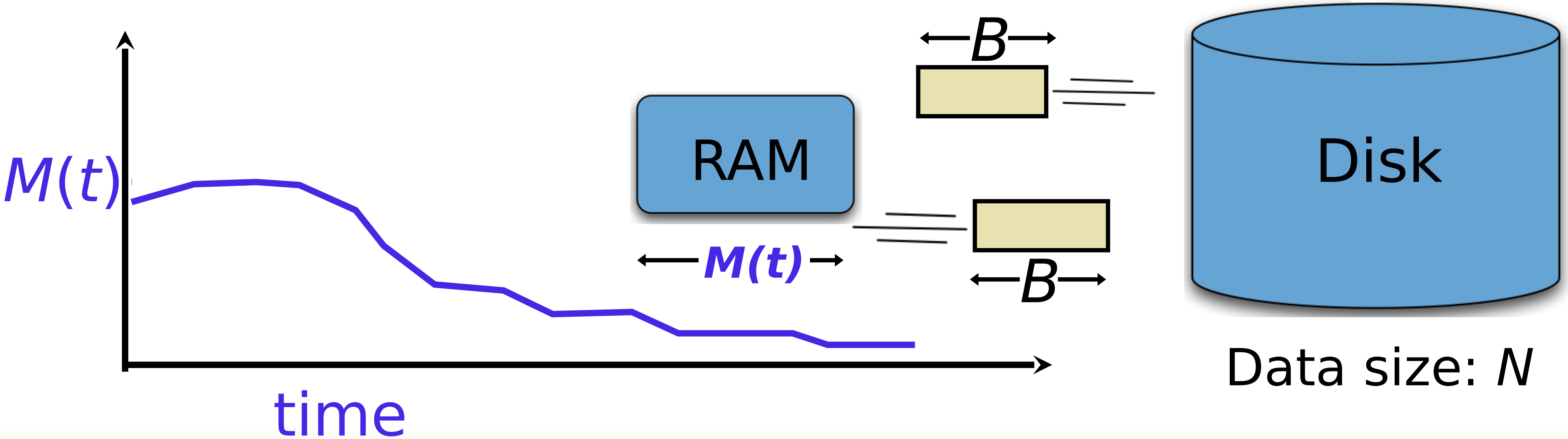
Cache-Adaptive Model

Generalizes Disk Access Machine (DAM) model [Aggarwal+Vitter '88].

- Data is transferred in blocks between RAM and disk.
- Performance is measured in terms of block transfers.

Now size of internal memory is a function of time.

- Can change arbitrarily
- Can change without advance notice



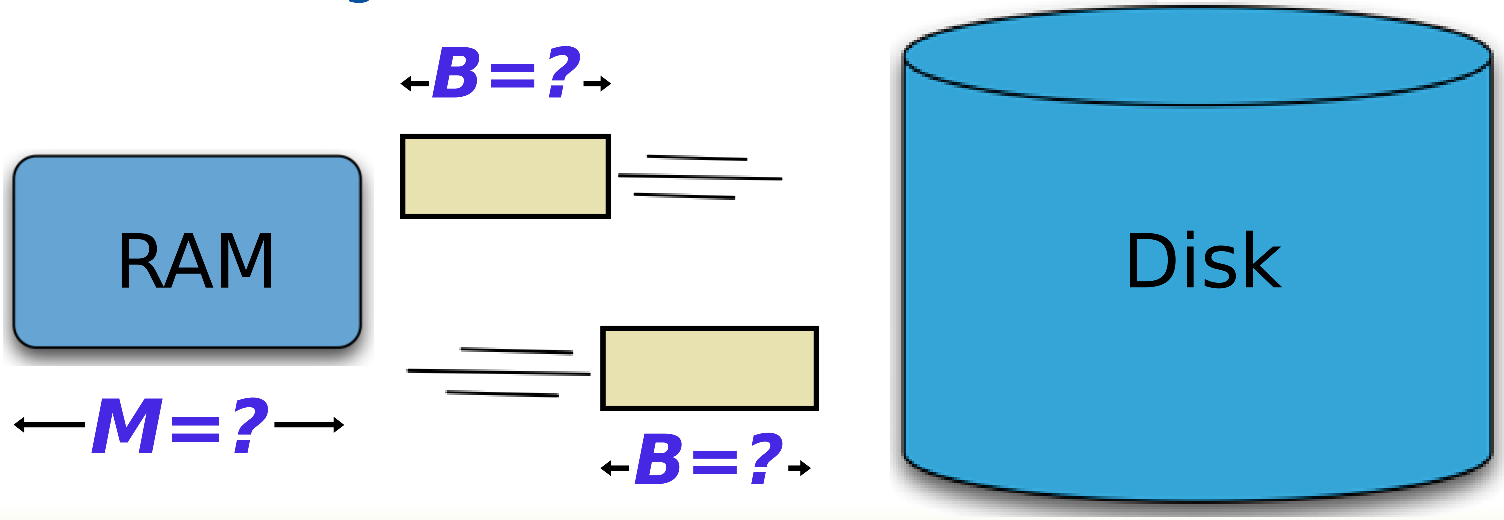
Cache-Oblivious Algorithms [Frigo, Leiserson, Prokop, Ramachandran '99]

Ideal-cache model: DAM model + automatic paging

- Contents of cache are managed by a separate paging algorithm.
- Time bounds are parameterized by ***B***, ***M***, ***N***.
- Goal: Minimize # of block transfers ≈ time.

Beautiful restriction:

- Parameters ***B***, ***M*** are unknown to the algorithm or coder.
- An optimal CO algorithm is universal for all ***B***, ***M***, ***N***.



Example: Recursive Matrix Multiplication is Cache-Oblivious

$N \times N$ matrix multiplication: 8 multiply-adds of $N/2 \times N/2$ matrices:

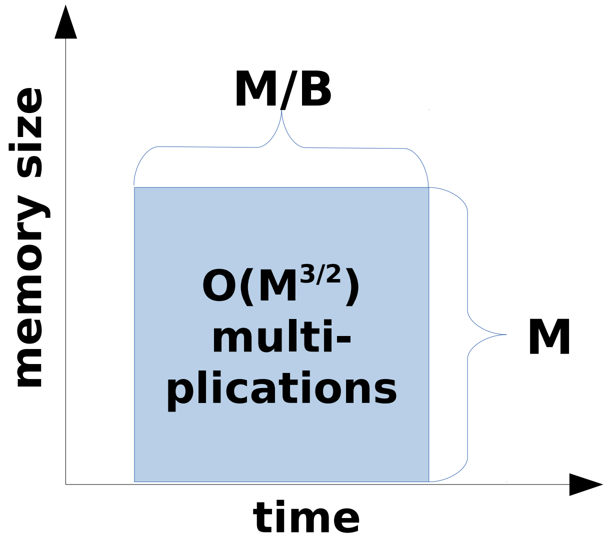
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix} + \begin{bmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{bmatrix}$$

$$\begin{aligned} T(N) &= \begin{cases} O(N^2/B) & \text{if } N^2 = O(M) \\ 8T(N/2) + O(N^2/B) & \text{otherwise} \end{cases} \\ &= O\left(\frac{N^3}{B\sqrt{M}}\right) \end{aligned}$$

Stony Brook University

Proving Algorithms Optimal in DAM Model: Progress Bounds

A ***progress bound*** $\rho(M)$ upper-bounds the amount of useful work that any algorithm can accomplish given M memory and M/B I/Os.



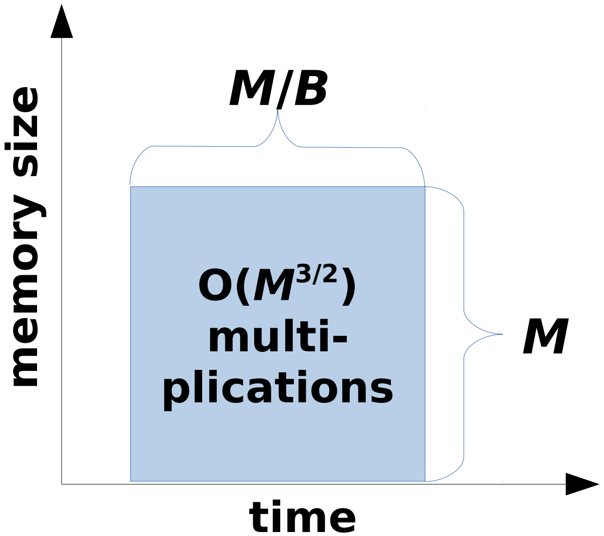
$$\rho(M) = O\left(M^{3/2}\right)$$

$$R(N) = O\left(N^{3}\right)$$

A ***progress requirement function*** $R(N)$ lower bounds the amount of work required to solve all problems of size N.

Example: Hong and Kung's progress bound for matrix multiplication [Hong and Kung 81]

Why Recursive Matrix Multiply is Optimal in the DAM Model



$$\rho(M) = O\left(M^{3/2}\right)$$

$$R(N) = O\left(N^3\right)$$

So no algorithm can have running time less than

$$\Omega\left(\frac{R(N)}{\rho(M)} \times \frac{M}{B}\right) = \Omega\left(\frac{N^3}{M^{3/2}} \times \frac{M}{B}\right) = \Omega\left(\frac{N^3}{B\sqrt{M}}\right)$$

CO matrix multiply running time: $T(N) = O\left(\dfrac{N^3}{B\sqrt{M}}\right)$

What Can Go Wrong in the CA Model?

$A * B$

$$R_1 = \begin{pmatrix} A_{11} * B_{11} & A_{11} * B_{12} \\ A_{21} * B_{11} & A_{21} * B_{12} \end{pmatrix}$$

$$R_2 = \begin{pmatrix} A_{12} * B_{21} & A_{12} * B_{22} \\ A_{22} * B_{21} & A_{22} * B_{22} \end{pmatrix}$$

8 recursive calls

return $R_1 + R_2$ — linear scan

No matter how much memory is available.

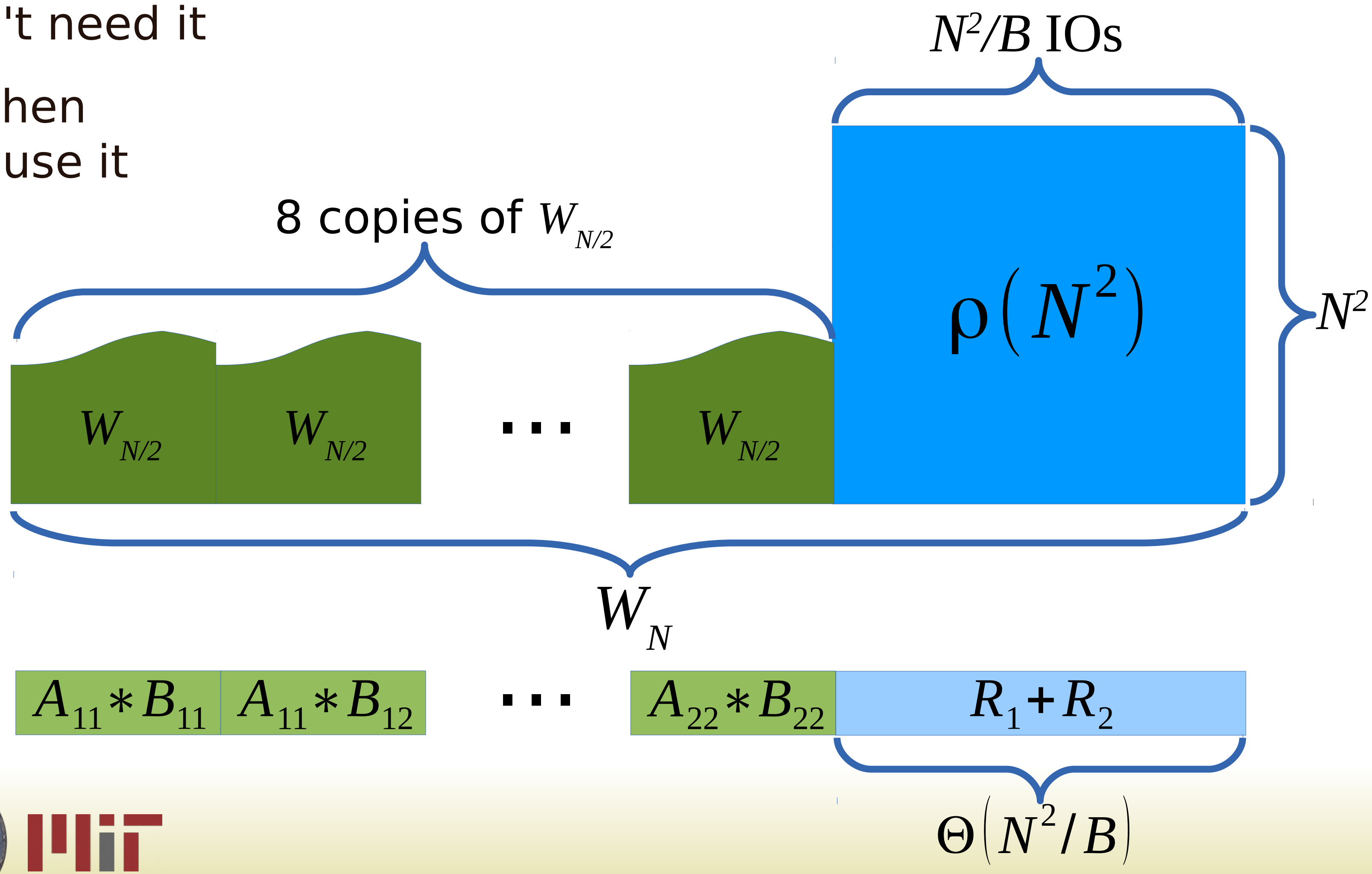
$\Theta\left(\frac{N^2}{B}\right)$ I/Os

$A_{11} * B_{11}$	$A_{11} * B_{12}$	⋯	$A_{22} * B_{22}$	$\underbrace{R_1 + R_2}_{\Theta(N^2/B)}$

What Can Go Wrong in the CA Model?

We can recursively construct a “bad” profile W_N that

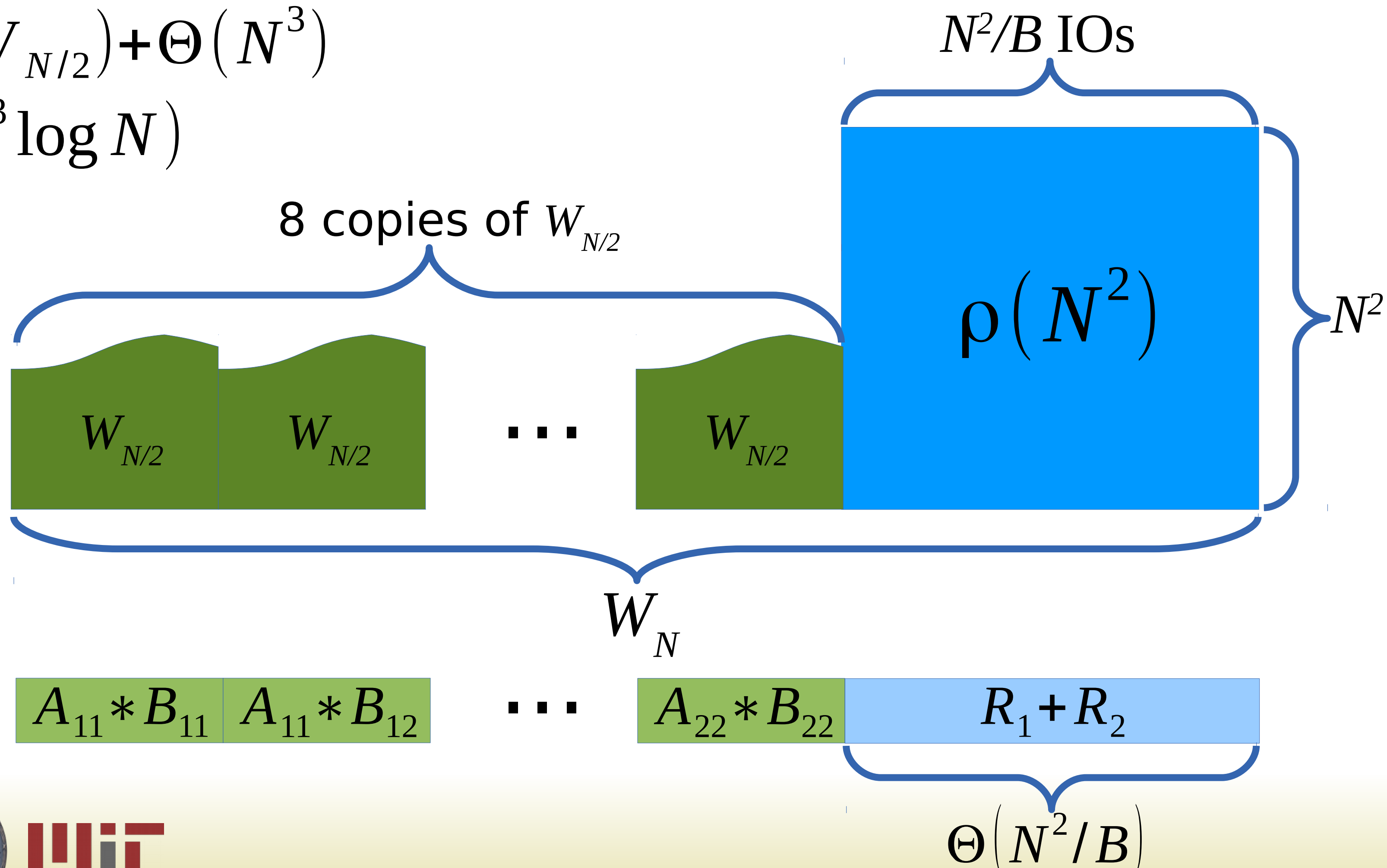
- Has lot's of memory when algorithm doesn't need it
- Little memory when algorithm could use it



What Can Go Wrong in the CA Model?

W_N supports a lot of progress:

$$\begin{aligned}\rho(W_N) &= 8\rho(W_{N/2}) + \Theta(\rho(N^2)) \\ &= 8\rho(W_{N/2}) + \Theta(N^3) \\ &= \Theta(N^3 \log N)\end{aligned}$$

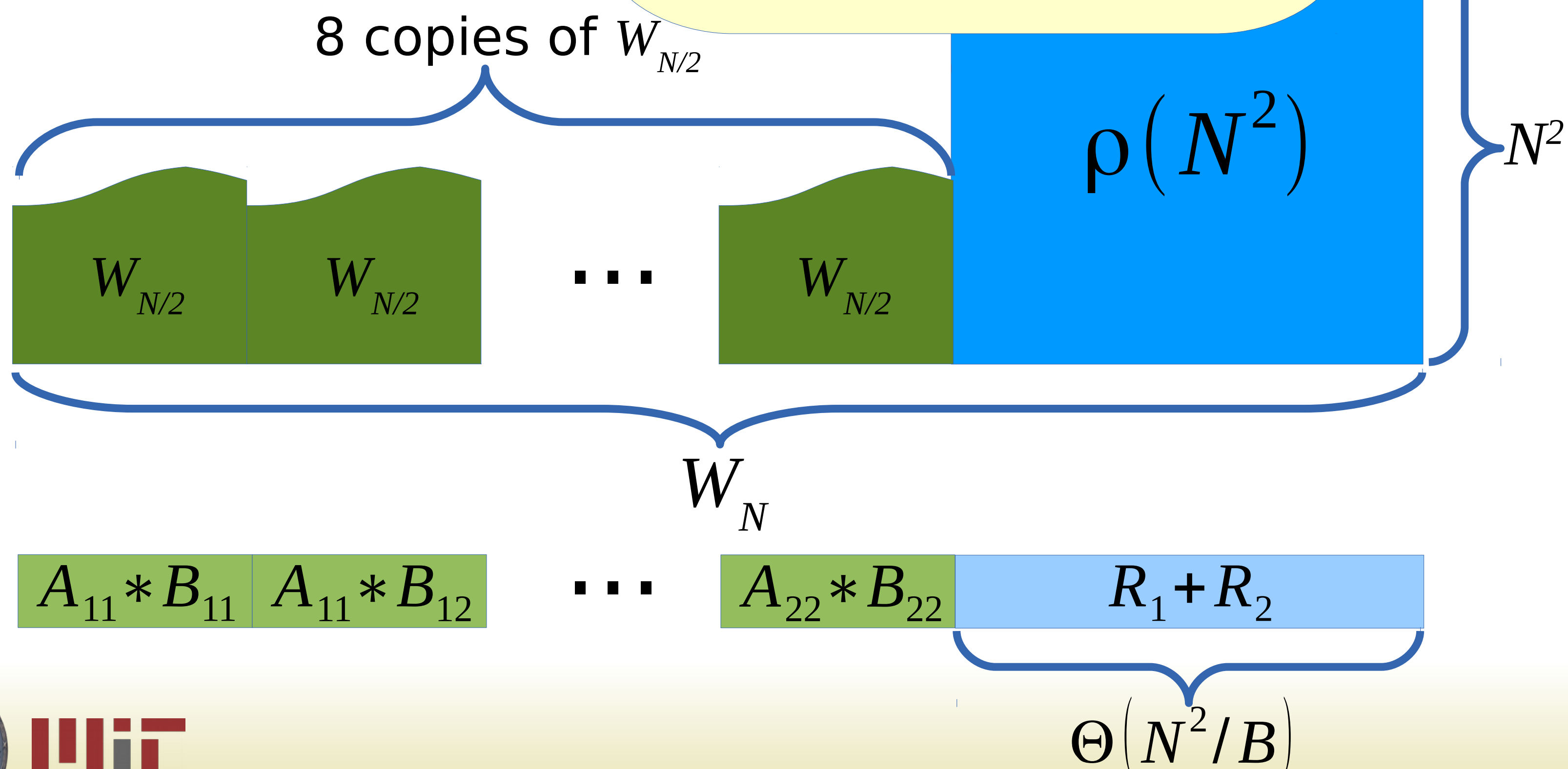


What Can Go Wrong in the CA Model?

W_N supports a lot of progress:

$$\begin{aligned}\rho(W_N) &= 8\rho(W_{N/2}) + \Theta(\rho(N^2)) \\ &= 8\rho(W_{N/2}) + \Theta(N^3) \\ &= \Theta(N^3 \log N)\end{aligned}$$

CO matrix multiply makes only $O(N^3)$ progress, so it is not optimal.



(Simplified) Recipe for Analyzing Cache Adaptivity

Write down recurrence relation for the algorithm:

$$T(N)=aT(N/b)+\Theta(N^c/B)$$

Derive new recurrence by replacing additive terms with progress bound ρ:

$$S(N)=aS(N/b)+\Theta(\rho(N^c))$$

If $S(N)=O(R(N))$, then the algorithm is optimally progressing.

This Recipe is General

Covers many different divide-and-conquer forms

- Master Theorem
- Akra-Bazzi
- Mutually recursive functions
- Plus others (e.g. cache-oblivious FFT)

Can answer several different questions

- Is an algorithm optimal?
- Is it not optimal?
- How far is it from optimal?

And it's easy!

- Just manipulating and solving recurrence relations

Conclusions

The CA model works.

- It is general enough to describe real systems.
- It is easy to work with.

Cache-oblivious algorithms are a good way to make CA algorithms.

- Many cache oblivious algorithms are CA.
- And are pretty close to optimal otherwise.