Parallel numerical algorithms: Course overview

Prof. Richard Vuduc

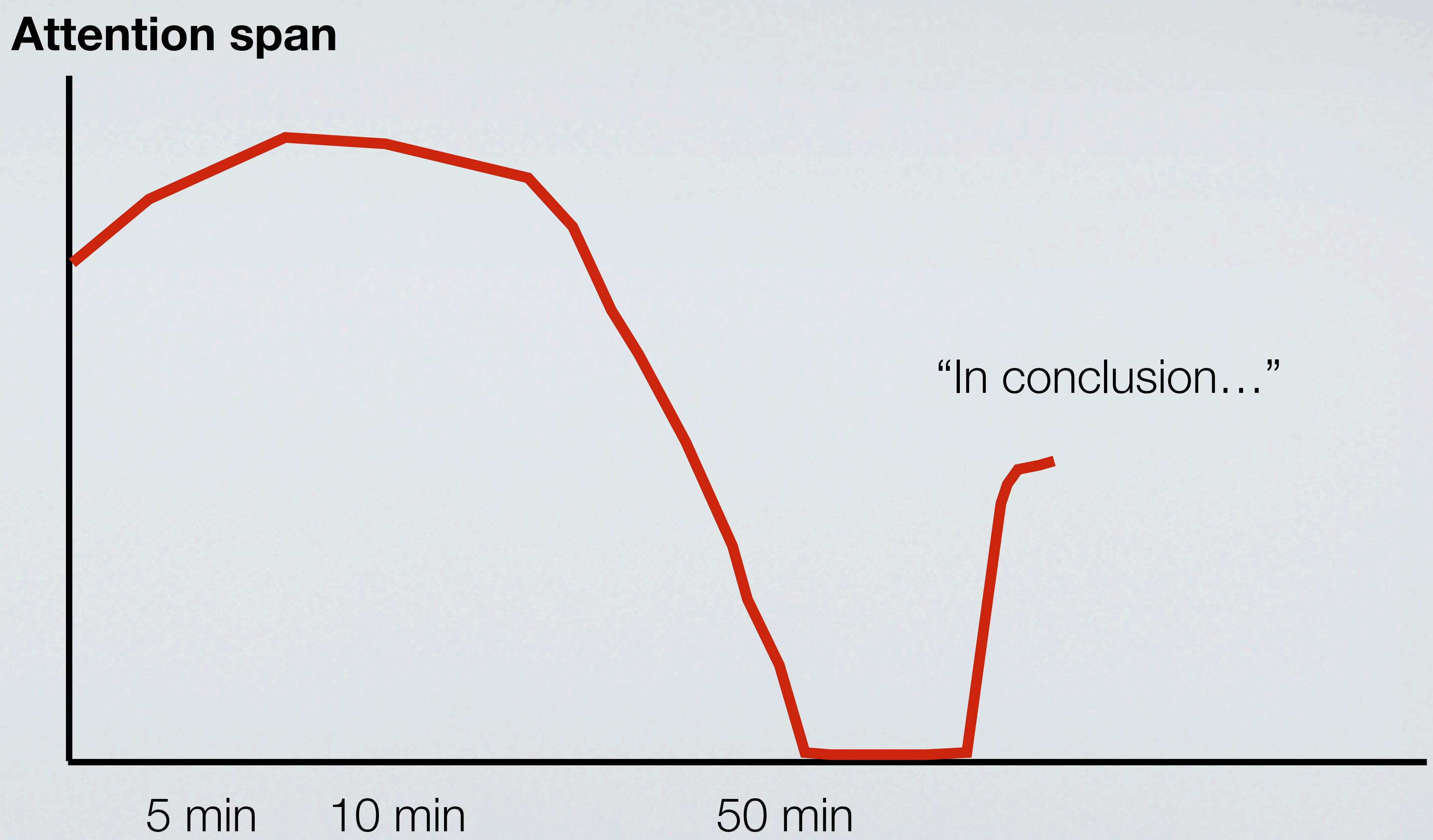
Georgia Institute of Technology

CSE/CS 8803 PNA, Spring 2008

[L.01] Tuesday, January 8, 2008

Sources for today's material

- CS 267 (Yelick @UCB)
- David Keyes
- Williams, *et al*. SC'07 (UCB)
- Higham, *Accuracy and Stability of Numerical Algorithms*



Lecture: Patterson's Law of Attention Span

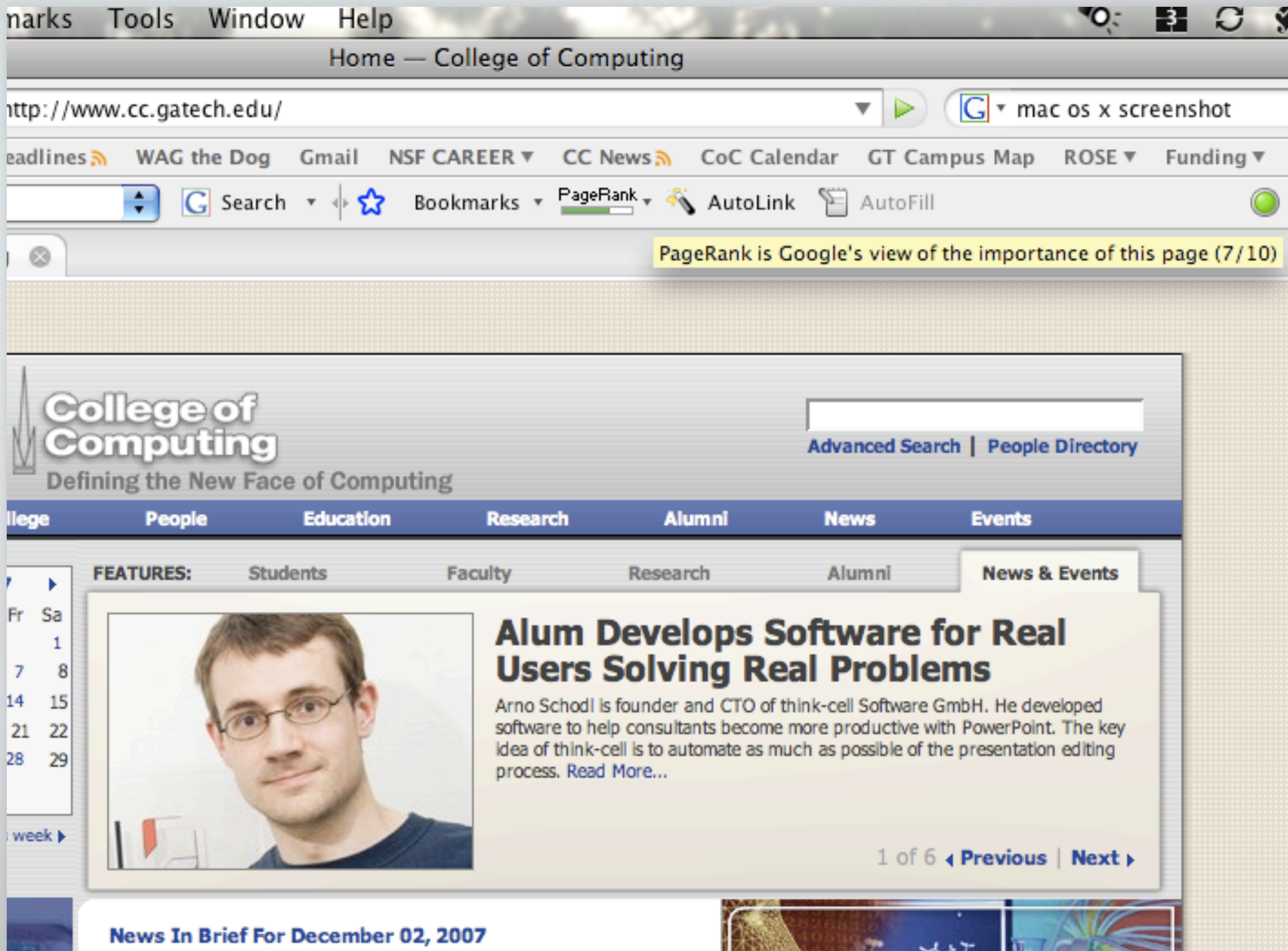
D. Patterson (UC Berkeley)

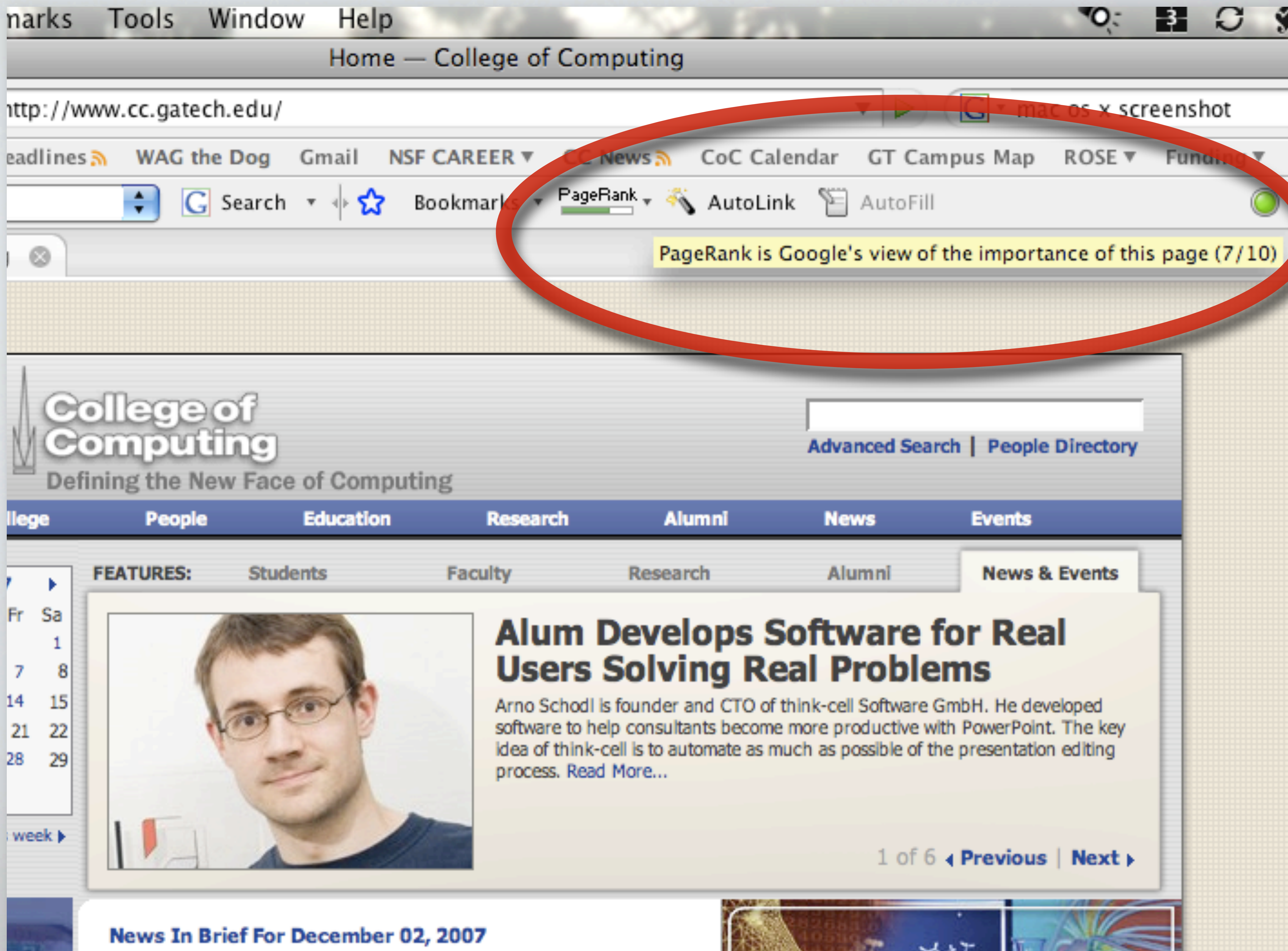
Why study PNA today?

Why study PNA today?

- Current and **future** apps are numerical, data-intensive
- Parallel hardware widely available
  - *Computing industry betting its future on it!* (next lecture)
- Need to program for parallelism and locality explicitly
- Algorithmic costs changed: “mops” not flops; “accuracy”
- “Winners” unclear

- Interaction of applications, algorithms, and systems


narks Tools Window Help
Home — College of Computing
ttp://www.cc.gatech.edu/
mac os x screenshot
eadlines WAG the Dog Gmail NSF CAREER CC News CoC Calendar GT Campus Map ROSE Funding
Search Bookmarks PageRank AutoLink AutoFill
PageRank is Google's view of the importance of this page (7/10)
College of Computing
Defining the New Face of Computing
Advanced Search | People Directory
llege People Education Research Alumni News Events
FEATURES: Students Faculty Research Alumni News & Events
Alum Develops Software for Real Users Solving Real Problems
Arno Schodl is founder and CTO of think-cell Software GmbH. He developed software to help consultants become more productive with PowerPoint. The key idea of think-cell is to automate as much as possible of the presentation editing process. Read More...
1 of 6 Previous | Next
News In Brief For December 02, 2007


narks Tools Window Help
Home — College of Computing
http://www.cc.gatech.edu/
mac os x screenshot
eadlines
WAG the Dog
Gmail
NSF CAREER
CC News
CoC Calendar
GT Campus Map
ROSE
Funding
Search
Bookmarks
PageRank
AutoLink
AutoFill
PageRank is Google's view of the importance of this page (7/10)
College of Computing
Defining the New Face of Computing
Advanced Search | People Directory
llege
People
Education
Research
Alumni
News
Events
FEATURES:
Students
Faculty
Research
Alumni
News & Events
Alum Develops Software for Real Users Solving Real Problems
Arno Schodl is founder and CTO of think-cell Software GmbH. He developed software to help consultants become more productive with PowerPoint. The key idea of think-cell is to automate as much as possible of the presentation editing process. Read More...
1 of 6 Previous | Next
week
News In Brief For December 02, 2007

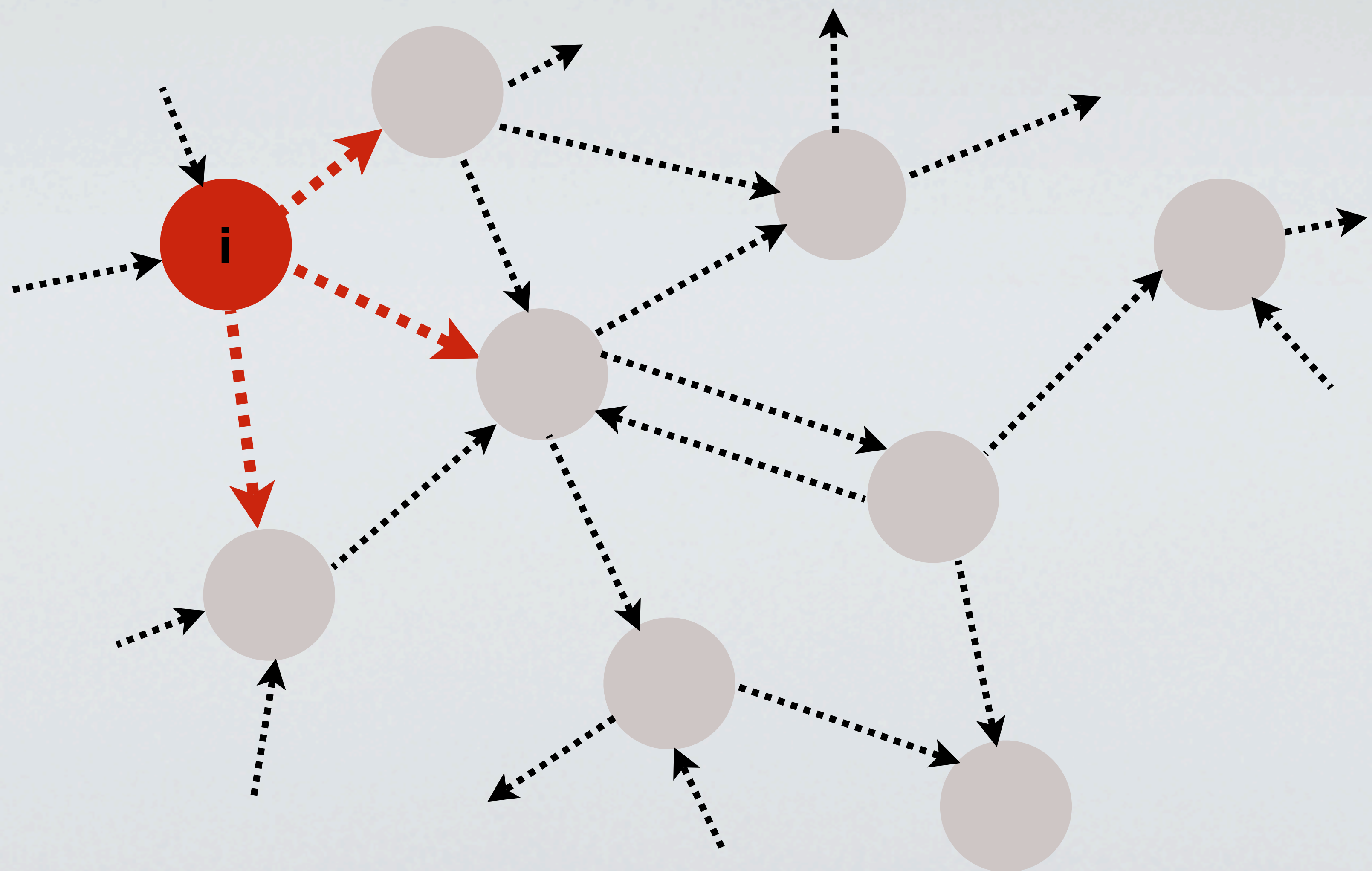
Application example: Google’s PageRank

Google’s web search procedure

- Step 1: Rank all web pages, given the web
  - Update every once in a while
- Step 2: Return list of matching pages in order by rank, given query
  - At “query-time”

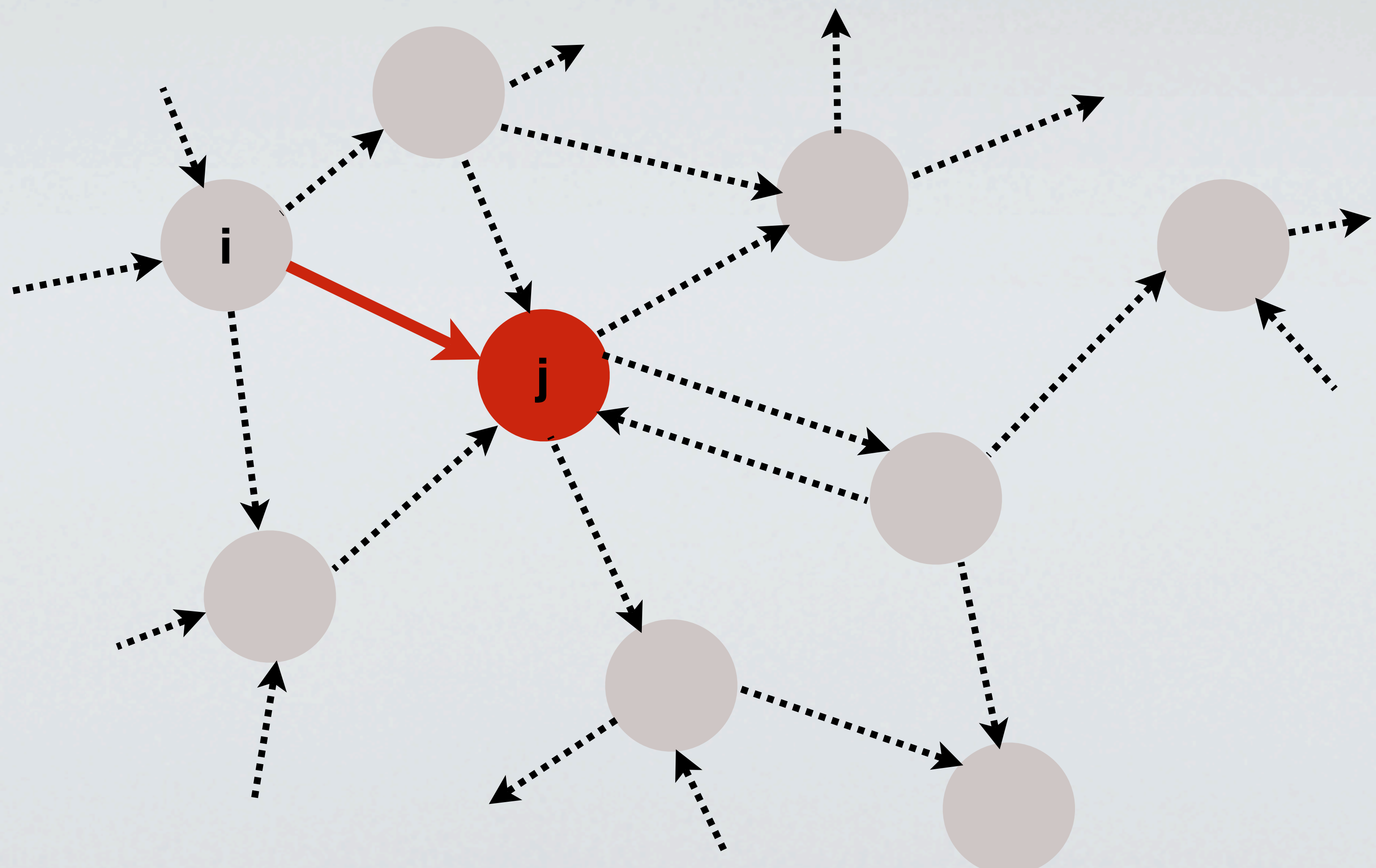
Google’s web search procedure

- Step 1: Rank all web pages, given the web
  - Update every once in a while
- Step 2: Return list of matching pages in order by rank, given query
  - At “query-time”
- **Compute ranking using a “random surfer” model**



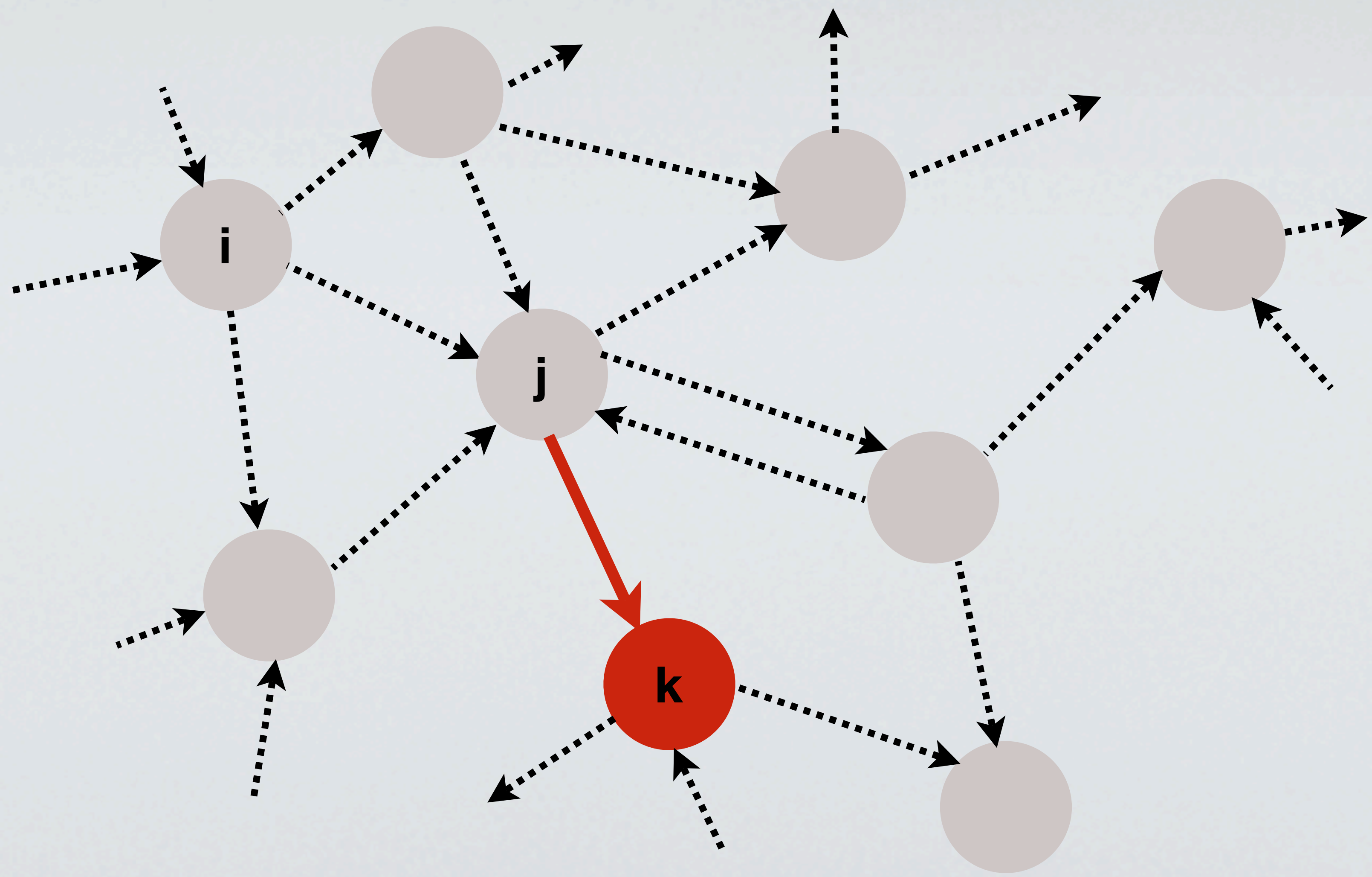
Start on page *i*.

A “random surfer” model.



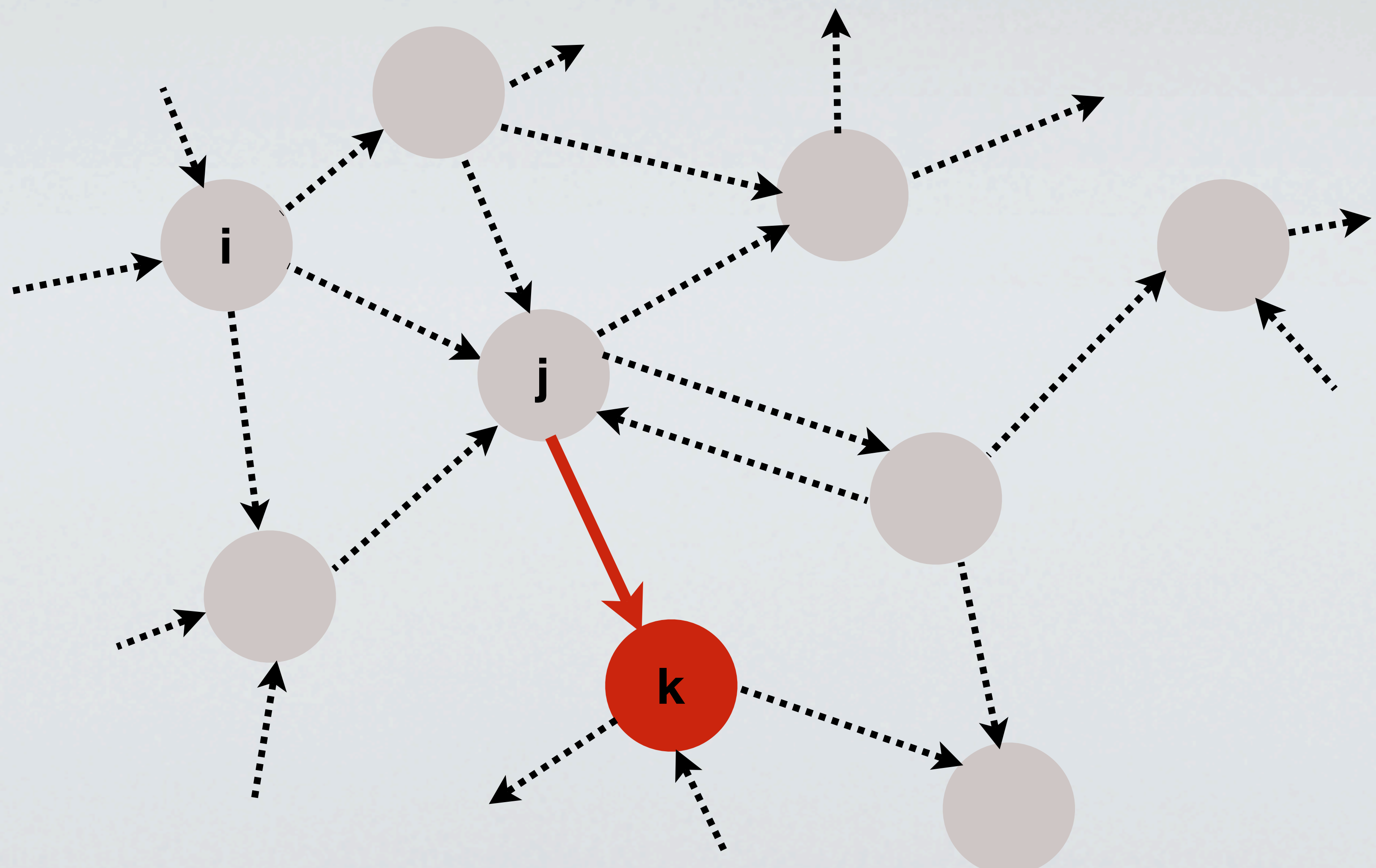
Jump to *j* with some probability.

A “random surfer” model.



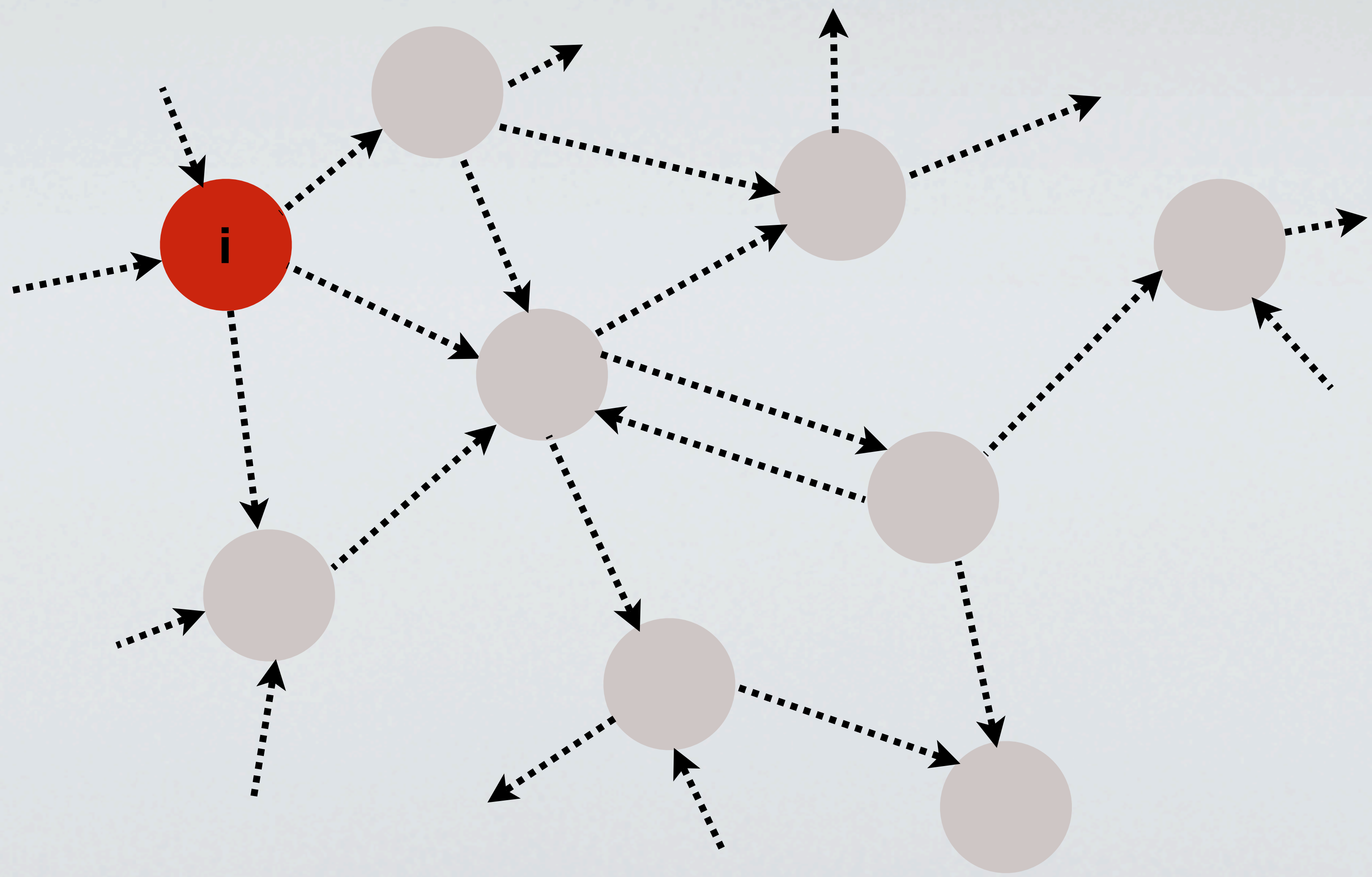
Repeat.

A “random surfer” model.



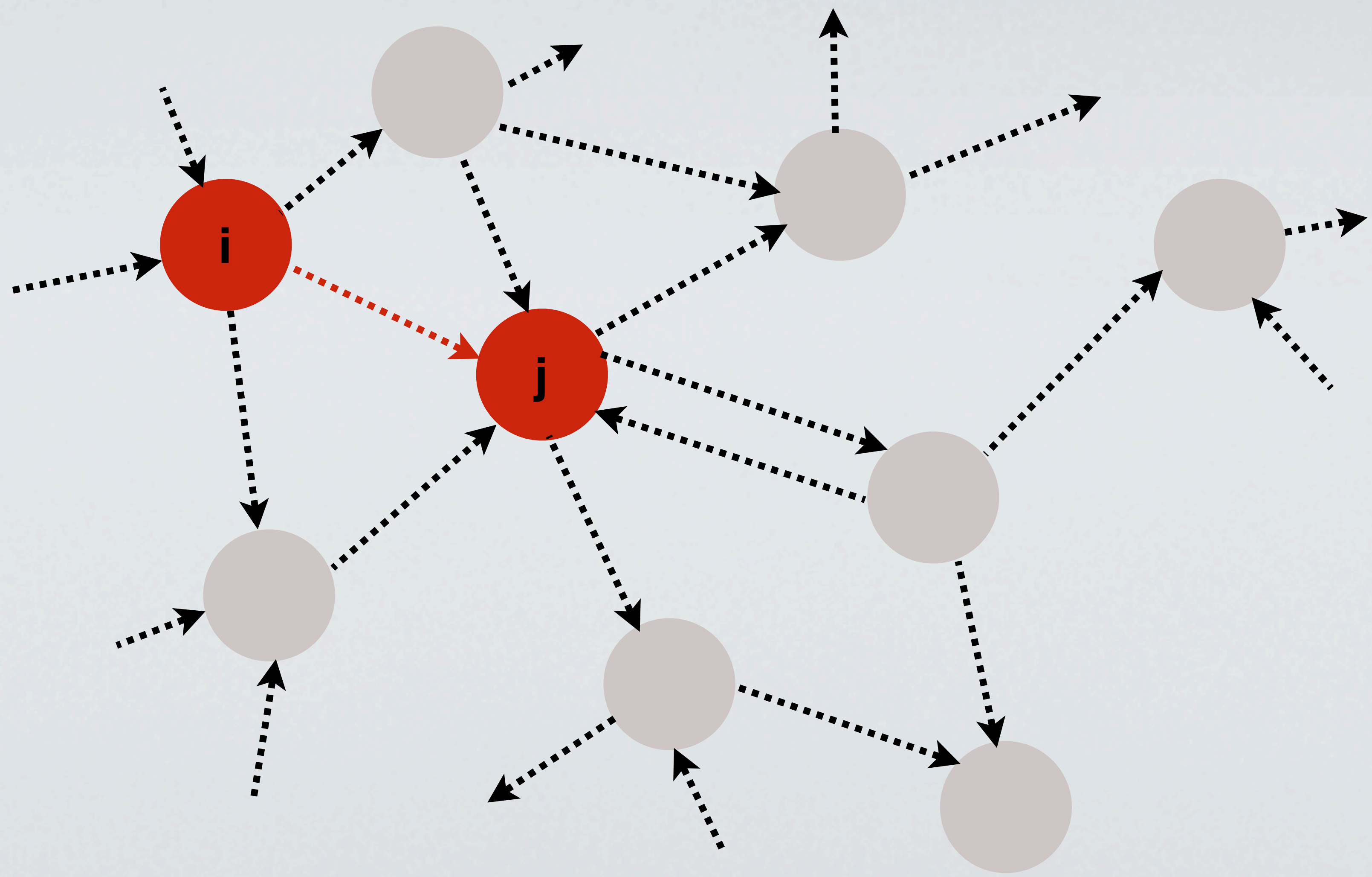
Where will the surfer end up?

A “random surfer” model.



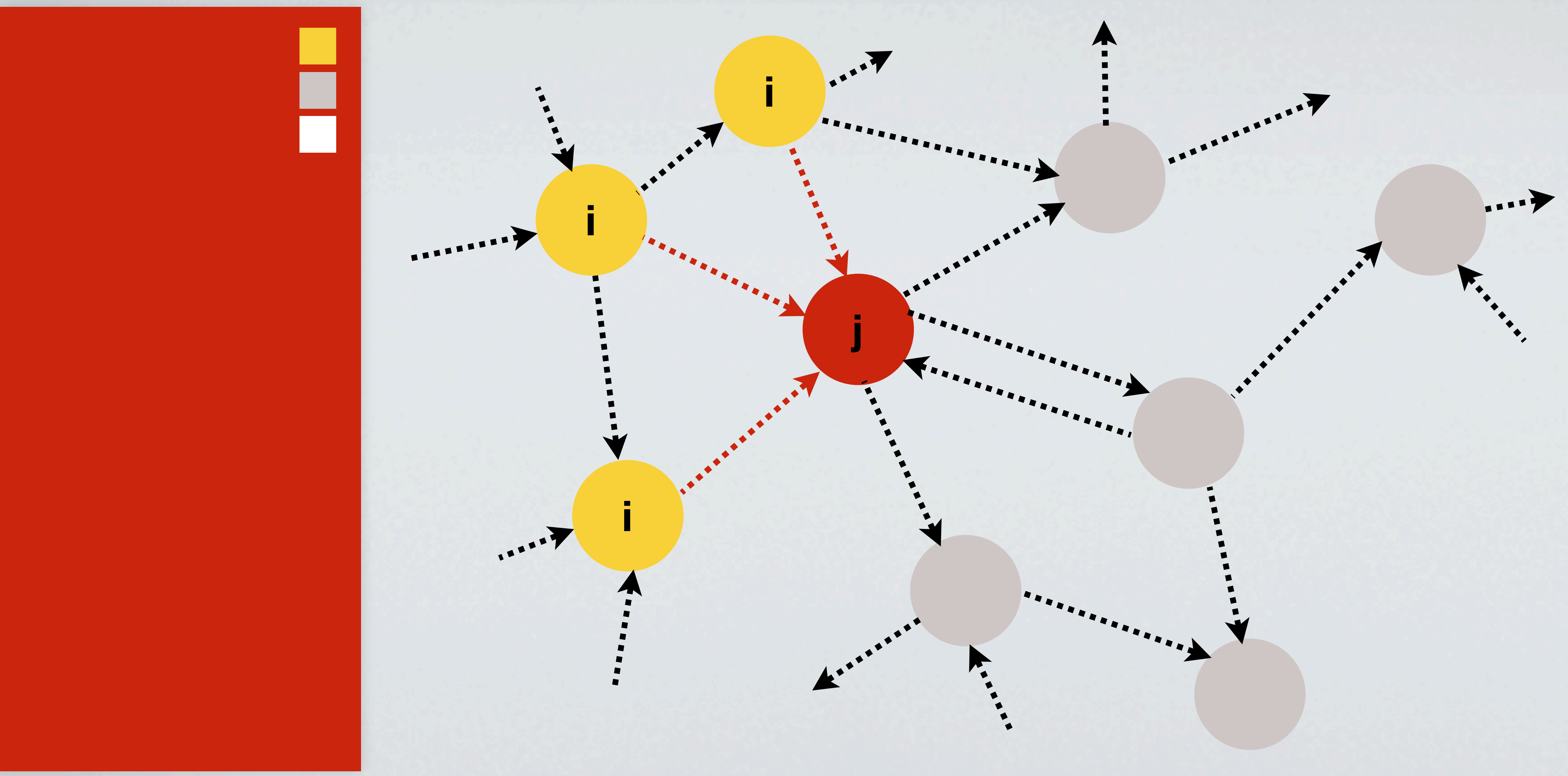
$$x_t(i)$$

Probability of being on page *i* at time *t*.



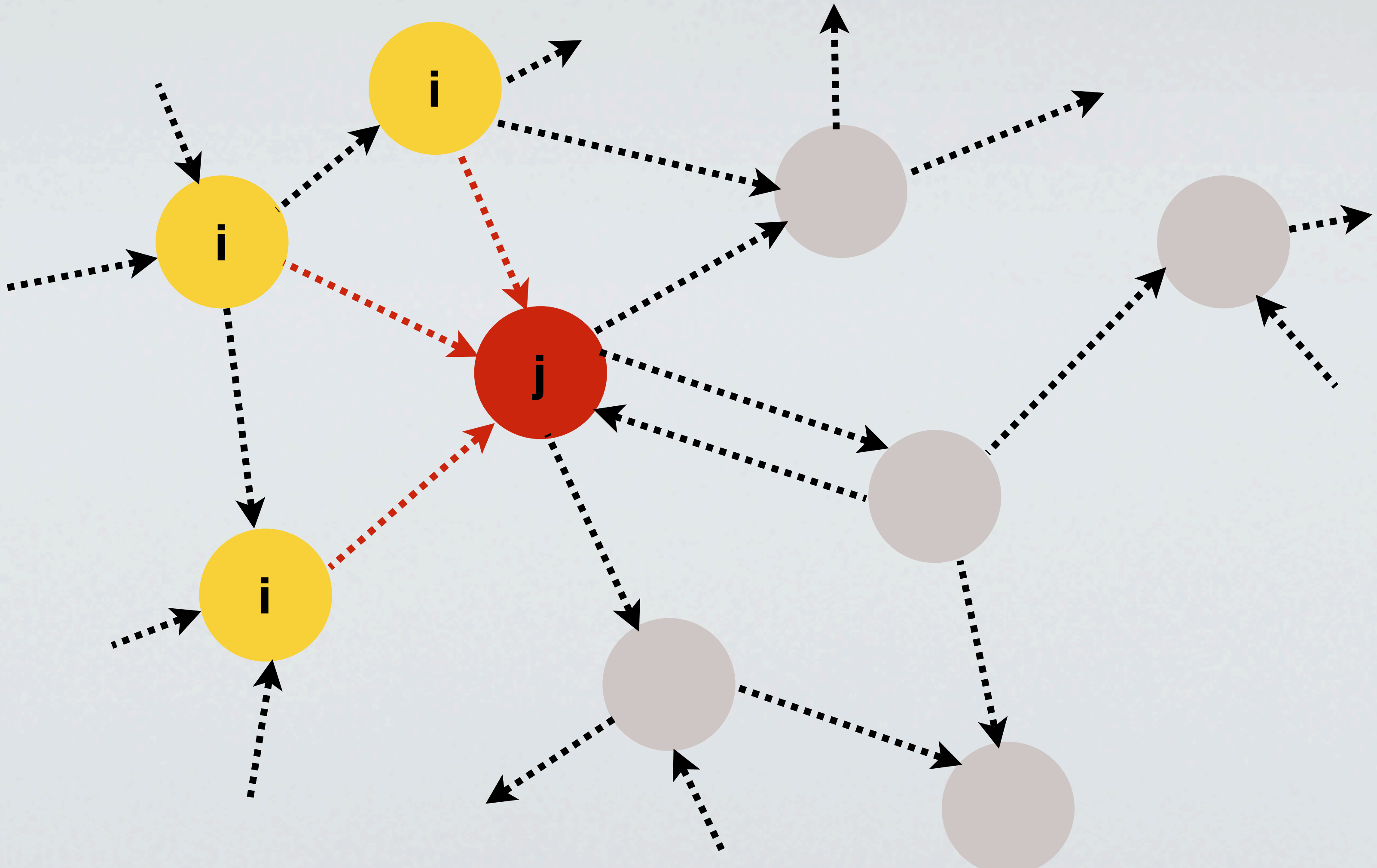
$$W(i, j)$$

Transition probability (=0 if no link).



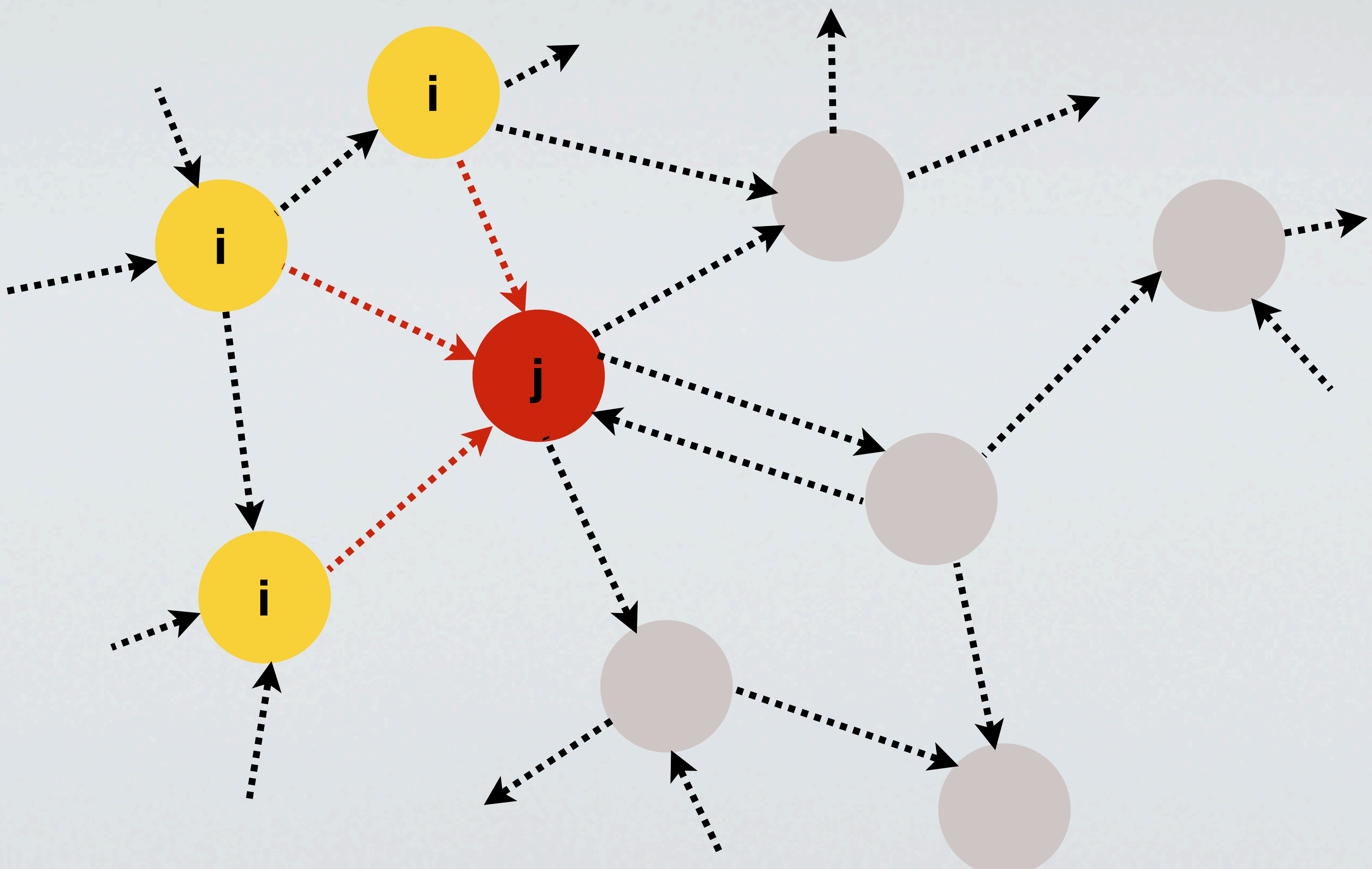
$$x_{t+1}(j) = \sum_{i=1}^{n} x_t(i) \cdot W(i,j)$$

Probability of being on page *j* at time *t+1*, with transition probability *W(i,j)*.



$$x_{t+1} = W^T \cdot x_t$$

Repeat **matrix-vector multiply** until convergence.



$$
\begin{aligned}
x_{t+1} &= W^T \cdot x_t = (W^T)^2 \cdot x_{t-1} = \cdots \\
&= (W^T)^{t+1} \cdot x_0
\end{aligned}
$$

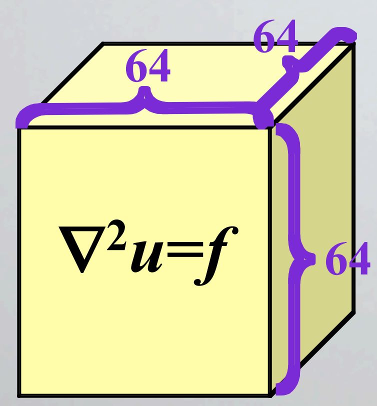
“Power method” for computing the principal eigenvector of W^T.

Applications and architectures

Units of measurement

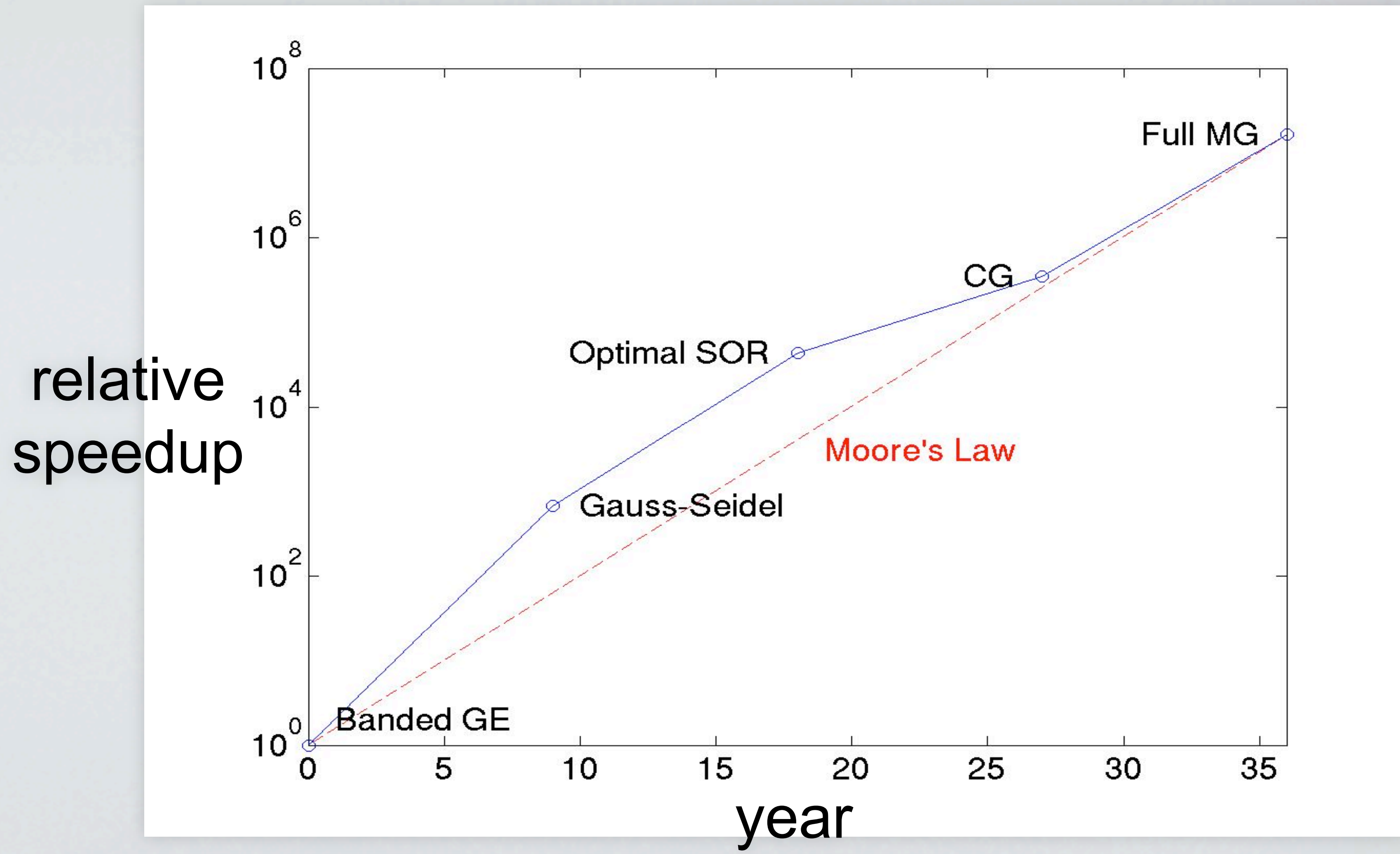
- “Flop”: Floating-point operation
- “Flop/s”: Flops per second
- “Bytes”: Size of data (double-precision float is 8 bytes)
- Today’s peak speeds
  - Single processor core ~5 Gflop/s
  - STI-Cell (8-core) ~15 Gflop/s
  - NVIDIA Tesla (128-core) ~500 Gflop/s
- Note: Achievable speeds often < 10% of peak

Year	Method	Reference	Storage	Flops
1947	Gaussian Elimination	Von Neumann & Goldstine	n^5	n^7
1950	Optimal SOR	Young	n^3	$n^4 \log n$
1971	Conj. grad.	Reid	n^3	$n^{3.5} \log n$
1984	Full multigrid	Brandt	n^3	n^3



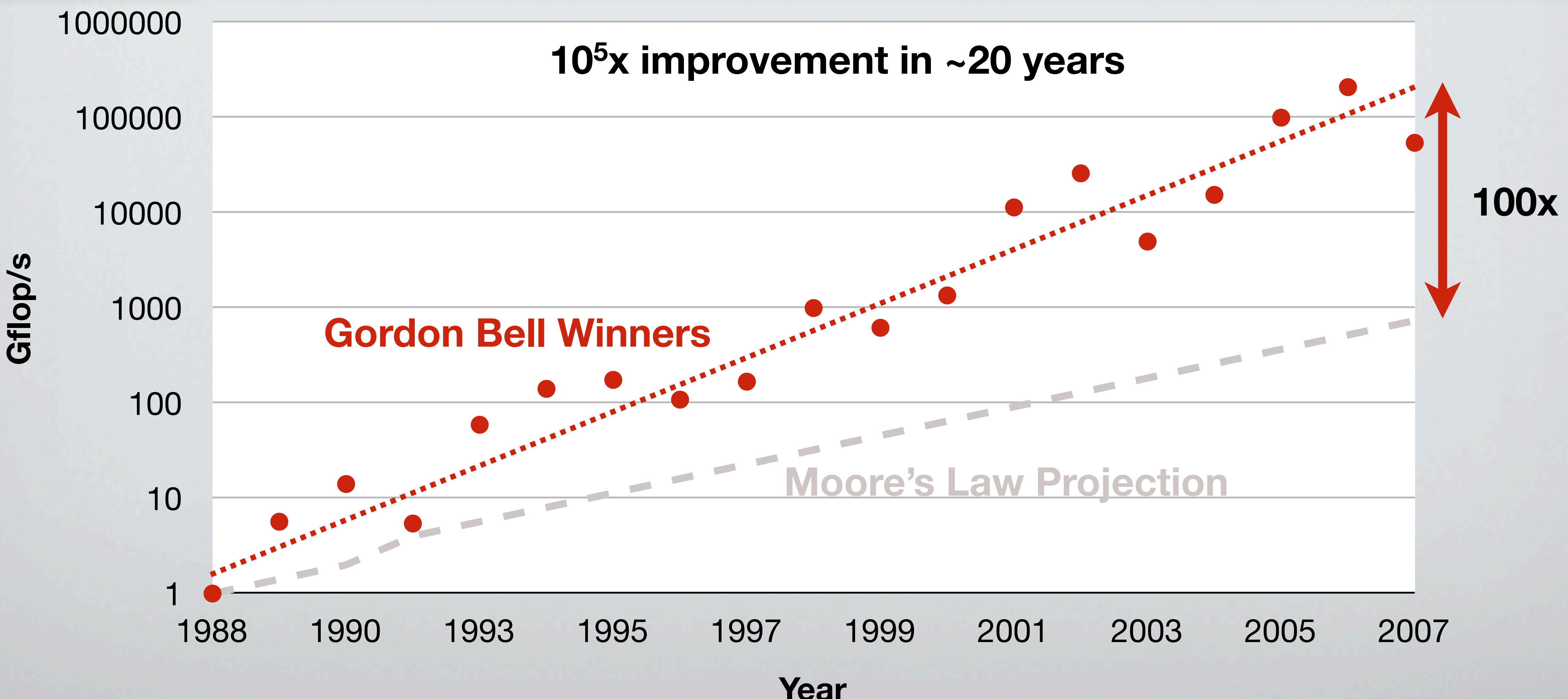
Algorithmic efficiency rivals architectural advances in scale.

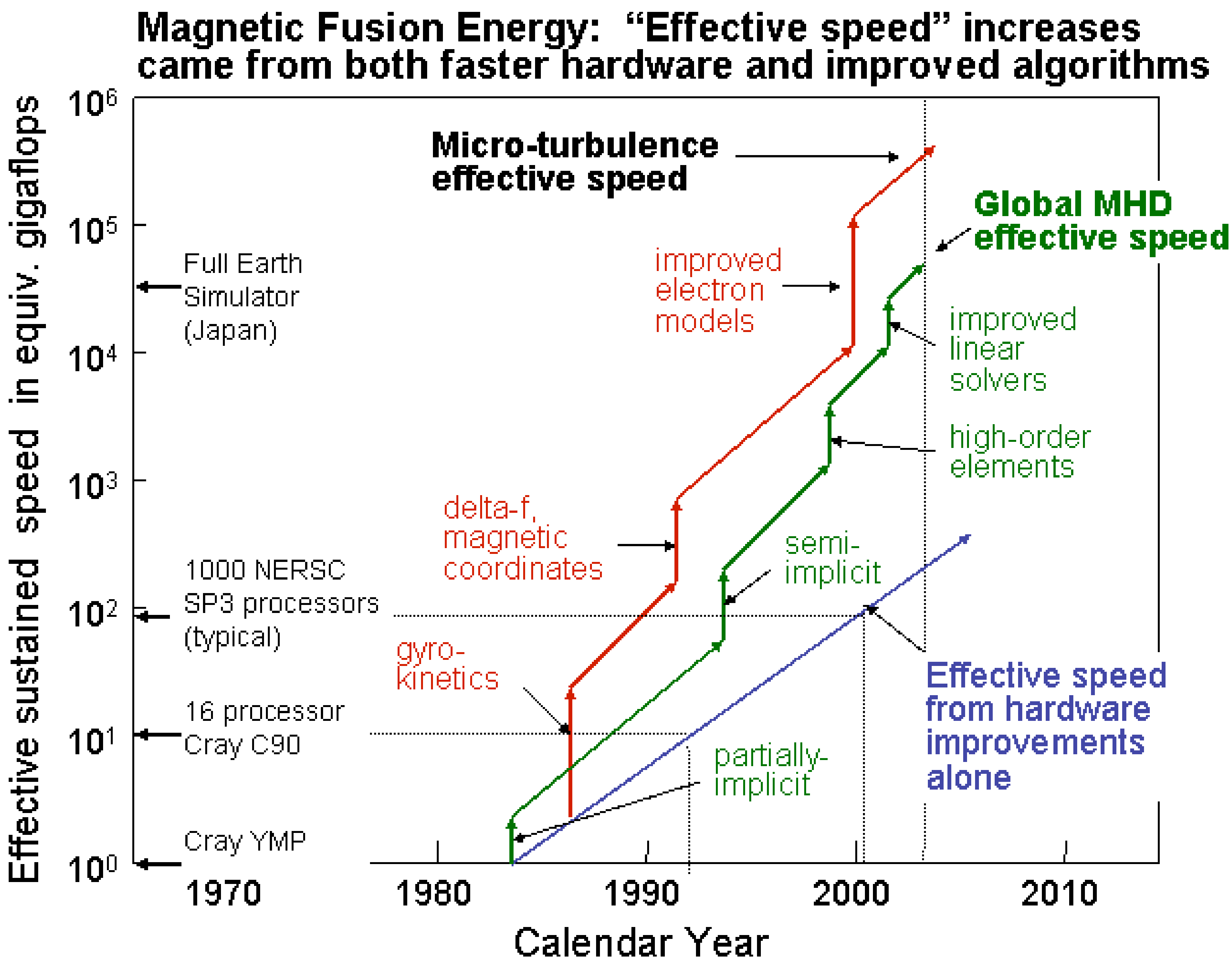
If n=64, flops reduced by ~16 M [6 mo. to 1 sec.]; Source: Keyes (2004)



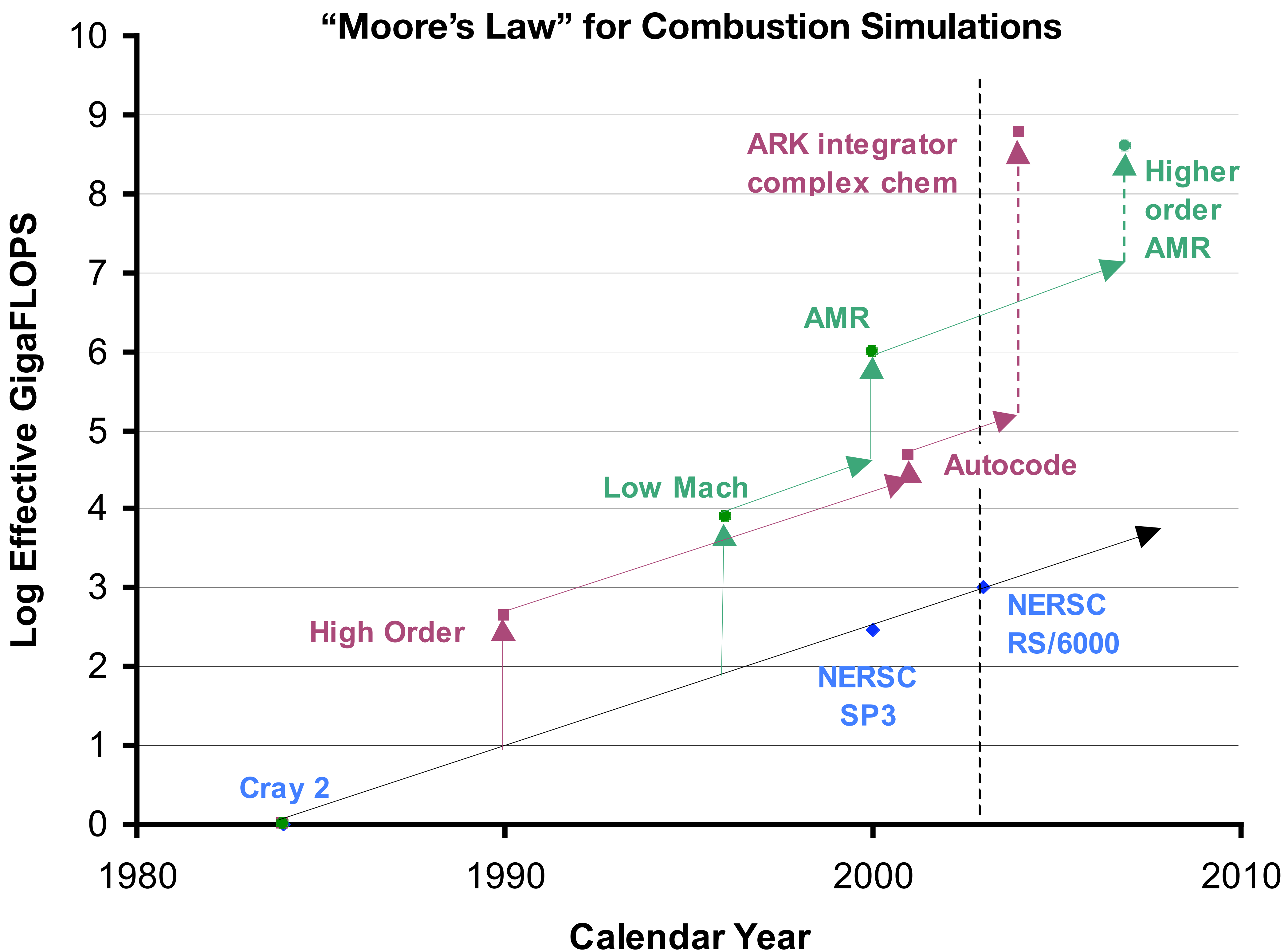
Algorithmic and architectural improvements go hand-in-hand.

Parallelism and algorithmic innovation outpace Moore's Law





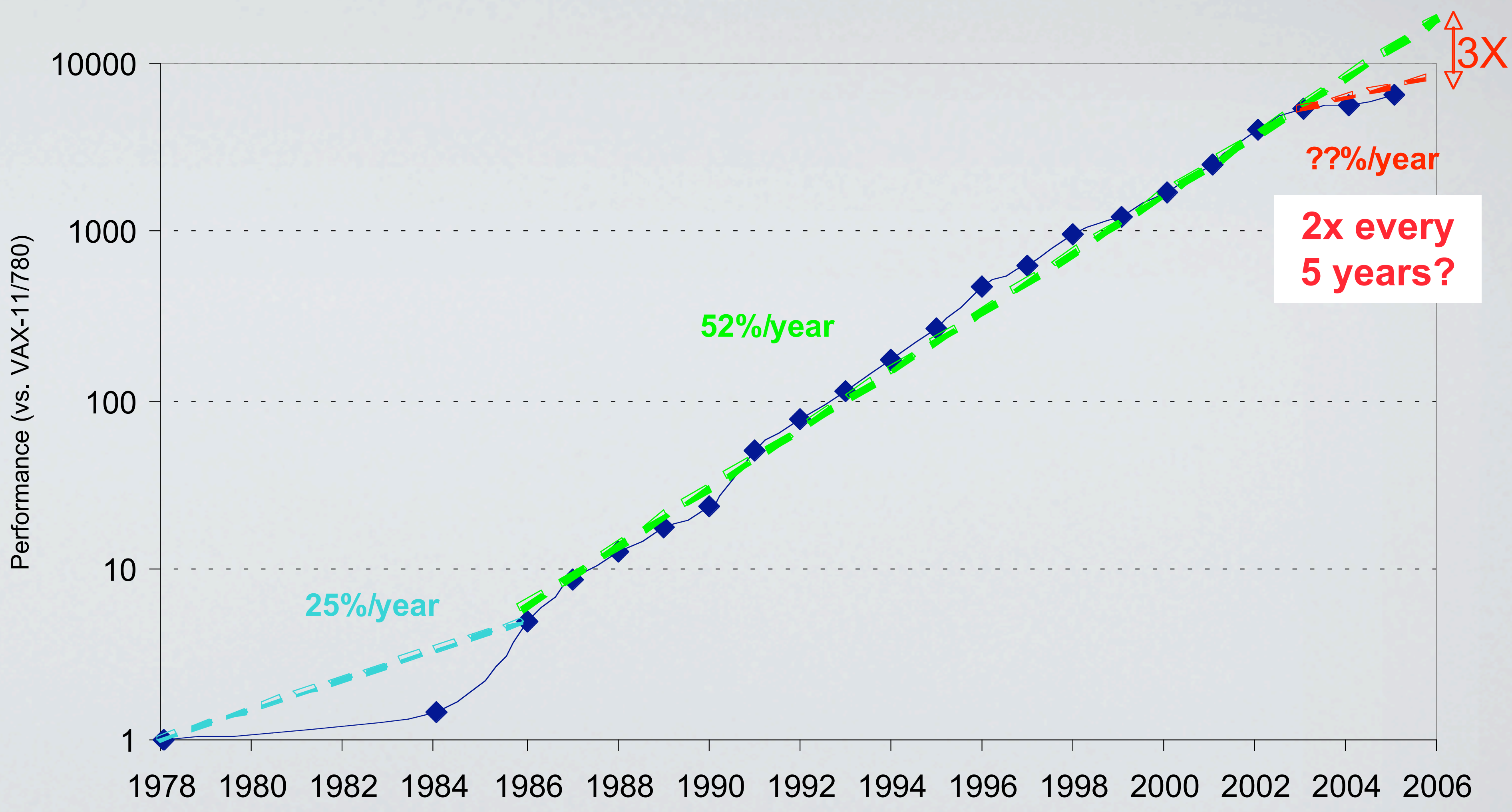
Source: SCaLeS report 2 (2004), *via* D. Keyes. http://pnl.gov/scales



Source: SCaLeS report 2 (2004), *via* D. Keyes. http://pnl.gov/scales

Intel, on their hardware strategy:

"We are dedicating all of our future product development to multicore designs. … This is a sea change in computing." (2005)



Uniprocessor performance not keeping up with Moore's law.

Source: Hennessey and Patterson, CA:AQA (4th edition), 2006

What’s so hard about parallel programming?

On p processors with fraction s serial work:

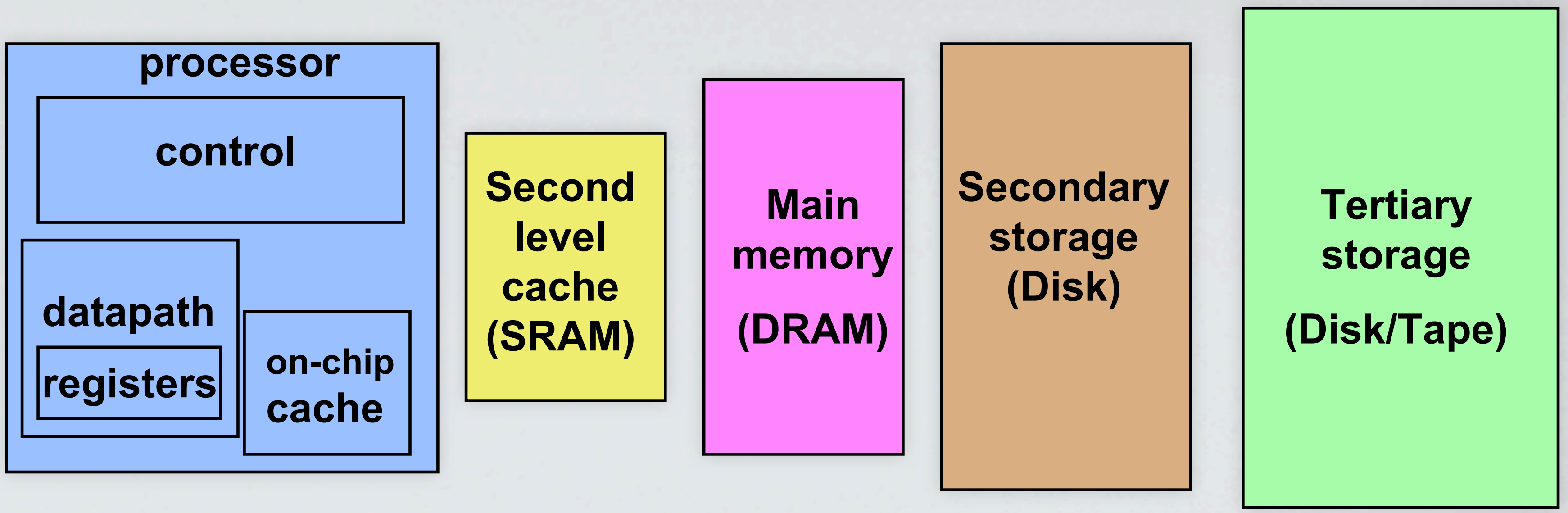
$$\begin{aligned}\text{Speedup}(p) &= \frac{\text{Time}(1)}{\text{Time}(p)} \\ &\leq \frac{1}{s + \frac{1-s}{p}} \\ &\leq \frac{1}{s}\end{aligned}$$

Finding enough parallelism.

Amdahl's Law: Maximum speedup limited by sequential part.

Parallelism incurs overheads.

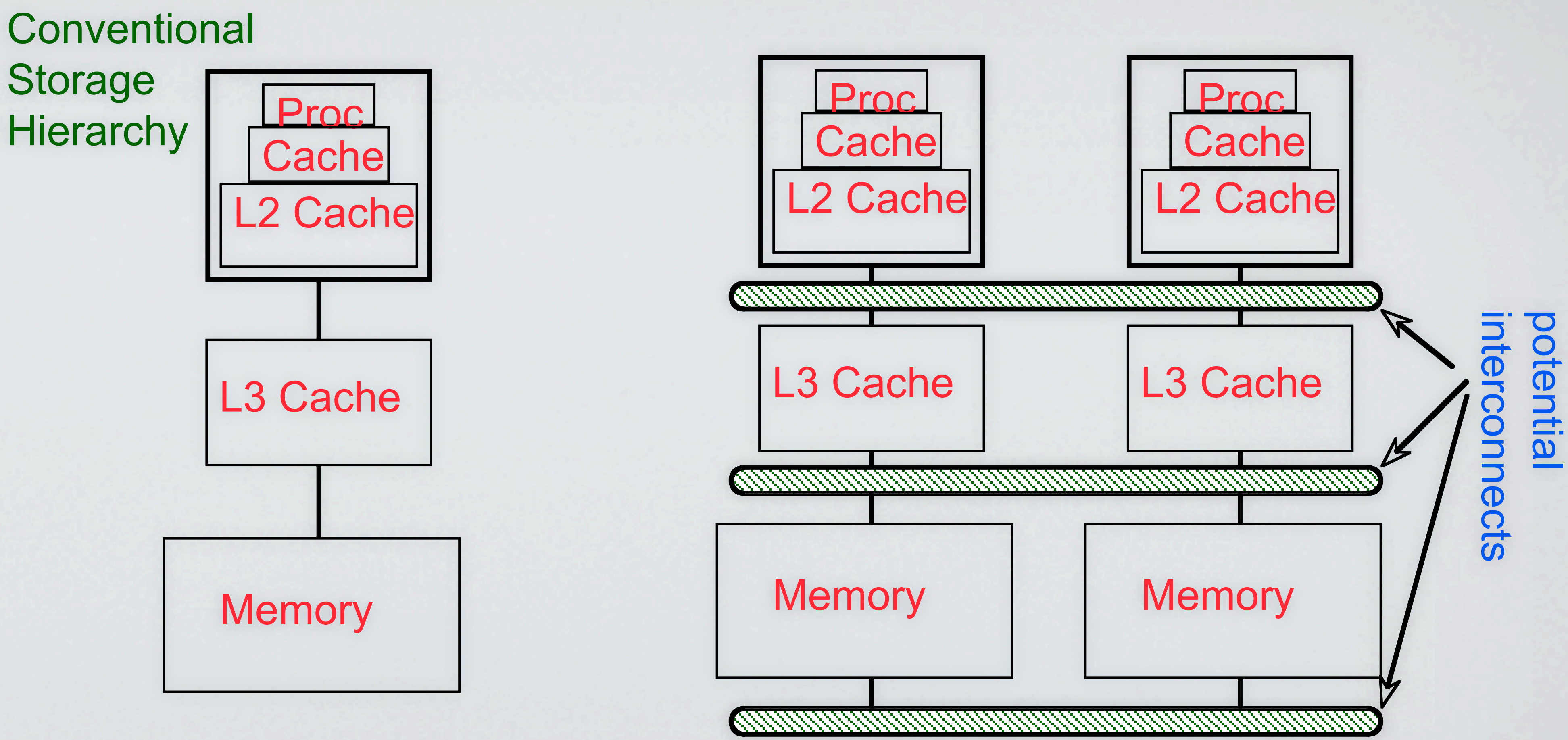
- Examples of overheads:
  - Cost of starting a thread or process
  - Cost of communicating shared data
  - Cost of synchronization
  - Cost of extra (redundant) computation
- Costs may be milliseconds, which is millions of flops
- Tradeoff: Granularity of task vs. amount of parallelism



Cost	1ns	10ns	100ns	10ms	10sec
Size	B	KB	MB	GB	TB

Memory hierarchies: Non-uniform memory access cost.

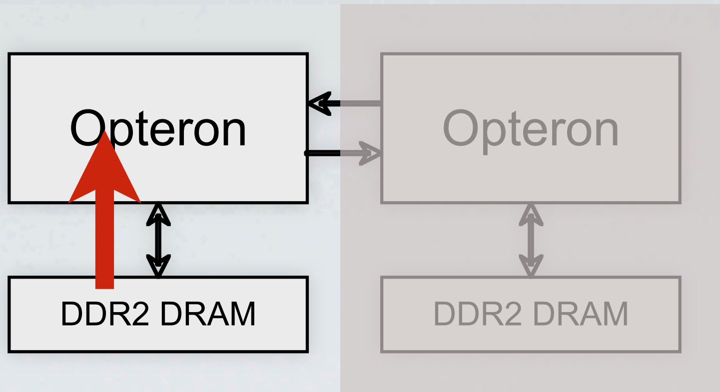
Better algorithms and implementations exploit locality.



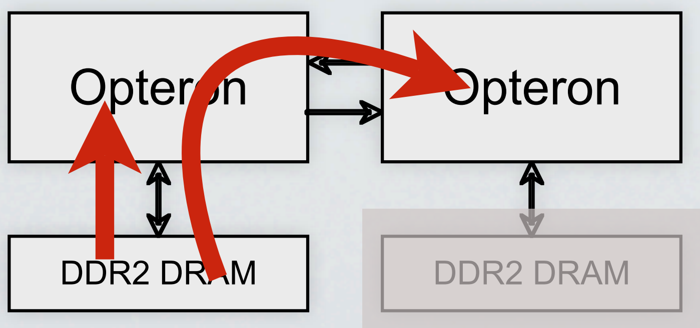
NUMA in the parallel setting.

Processors should minimize communication (remote data access).

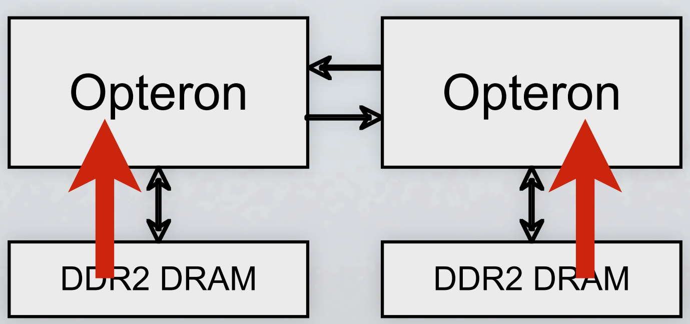
NUMA ⇒ Explicit data placement.



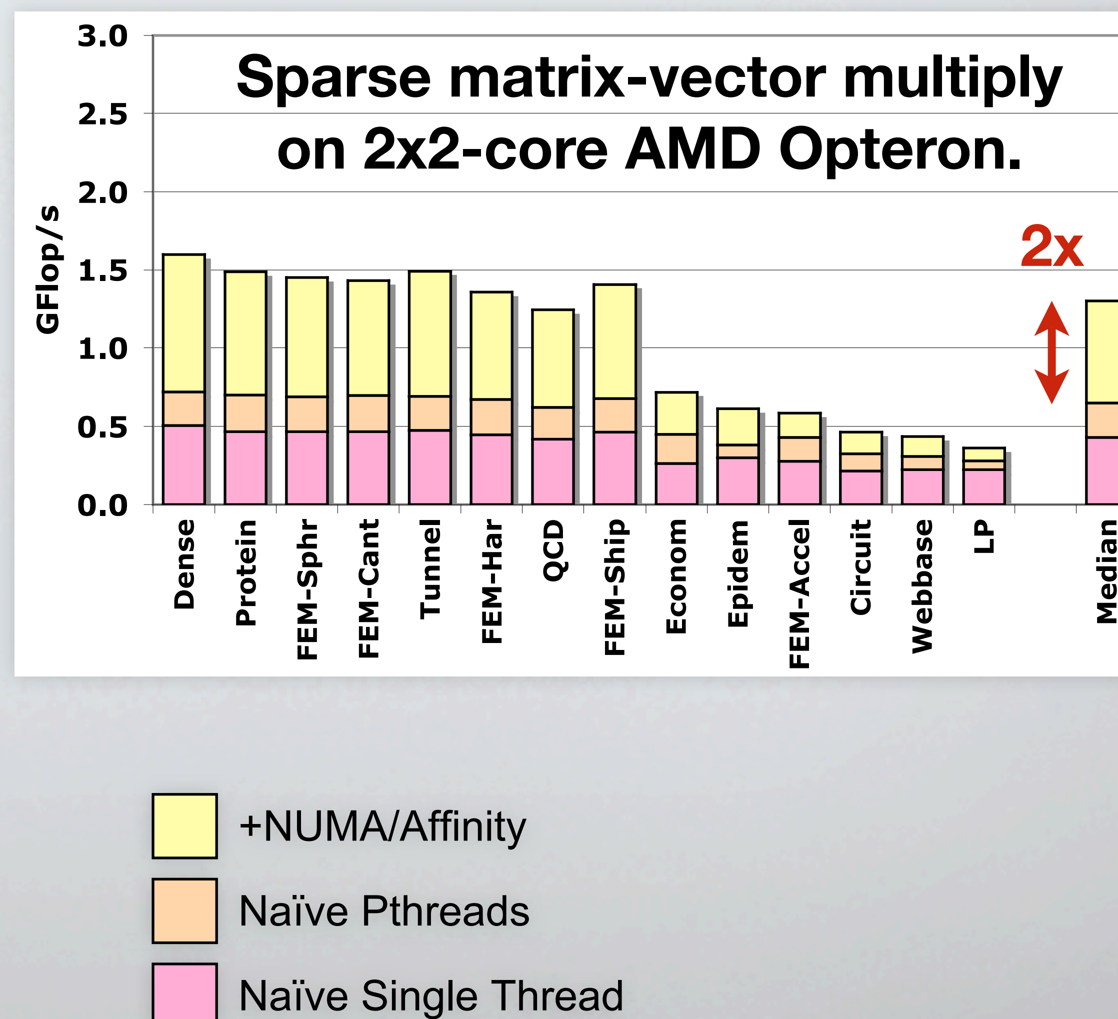
Single Thread



Multiple Threads,
One memory controller



Multiple Threads,
Both memory controllers



$$y = AA^T x$$

$$\begin{aligned} t &\leftarrow A^T x \\ y &\leftarrow A t \end{aligned}$$

For large *A*, "reads" *A* twice.

NUMA ⇒ Seek **algorithmic** reuse.

$$AA^T x = (a_1 \cdots a_n) \begin{pmatrix} a_1^T \\ \vdots \\ a_n^T \end{pmatrix} x = \sum_{i=1}^{n} a_i \left(a_i^T x\right)$$

Reorganize to improve reuse.

Algorithms and implementations must guard against load imbalance.

- Load imbalance: Subset of processors becomes idle
  - Insufficient parallelism
  - Unequally sized tasks
- Sources of imbalance
  - Local adaptation (adaptive mesh refinement)
  - Tree-structured computations
  - Fundamentally unstructured problem

Large-scale parallelism may raise numerical accuracy issues.

- Finite-ness of precision matters more for larger problems
  - Ill-conditioning
  - Round-off
- Example: Compute variance of data set using "two-pass" algorithm (right)
  - In single prec. ($\epsilon \sim 10^{-7}$), all digits lost when $n \sim 10$ million

$$
\begin{aligned}
\bar{x} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\sigma(x) &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2
\end{aligned}
$$

$$
\begin{aligned}
\text{Let } \hat{\sigma}(x) &= \textit{computed } \sigma(x), \\
\text{and } \epsilon &= \text{machine precision.} \\
\text{then:} &
\end{aligned}
$$

$$
\frac{\hat{\sigma}(x) - \sigma(x)}{\sigma(x)} \leq (n+3)\epsilon + O\left(\epsilon^2\right)
$$

Course organization

Student make-up of course?

- Mostly CS students, so will emphasize algorithm-to-machine mapping more than new-algorithm-development
- Work in teams of two and three; be interdisciplinary where possible!

Workload and grading

- Grading
  - 10% - Class participation and “scribe notes”
  - 24% - Two homework/programming assignments
  - 6% - Attend SIAM PP and write-up what you learned
  - **60% - Course project**
- All homework during first 8 weeks; last 8 weeks for your project!
- Collaboration encouraged, but don’t “cheat.”
- No textbook, but will supplement lectures with readings

Schedule of topics (approximate)

- Week 1: Overview and hardware trends
- Week 2: Sources of parallelism & locality; performance modeling
- Week 3: Basics of parallel programming [HW 1]
- Week 4: Structured grids; dense linear algebra
- Week 5-6: Dense and sparse linear algebra
- Week 7: FFT ; floating-point issues
- Week 8: Single-processor performance tuning [HW 2; project proposals]
  - Guest lecture by Hyesoon Kim on General-purpose GPU programming

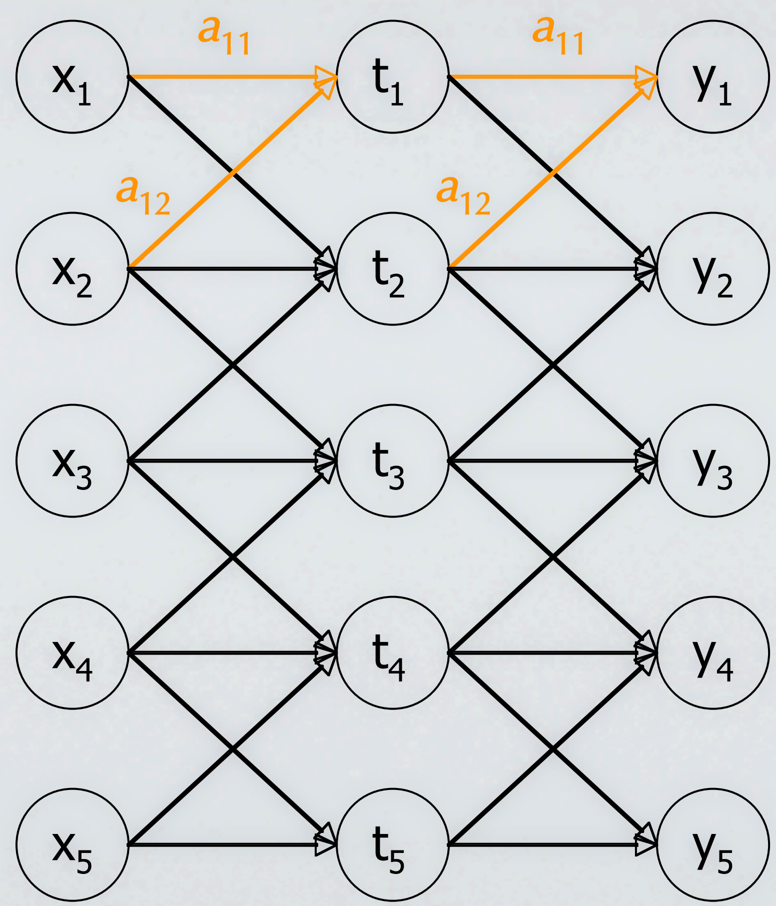
Schedule of topics (2)

- Week 9: Automatic performance tuning (autotuning)
- Week 10: Load balancing; SIAM PP
- Week 11-12: Particle methods; graph partitioning [Project checkpoint]
- Week 13: PDEs
- Week 14: Event-driven methods
- Week 15: Volunteer computing ; parallel languages research
- Week 16: Project presentations
- Project write-up due on “final exam” day

Homework #0: Complete course survey

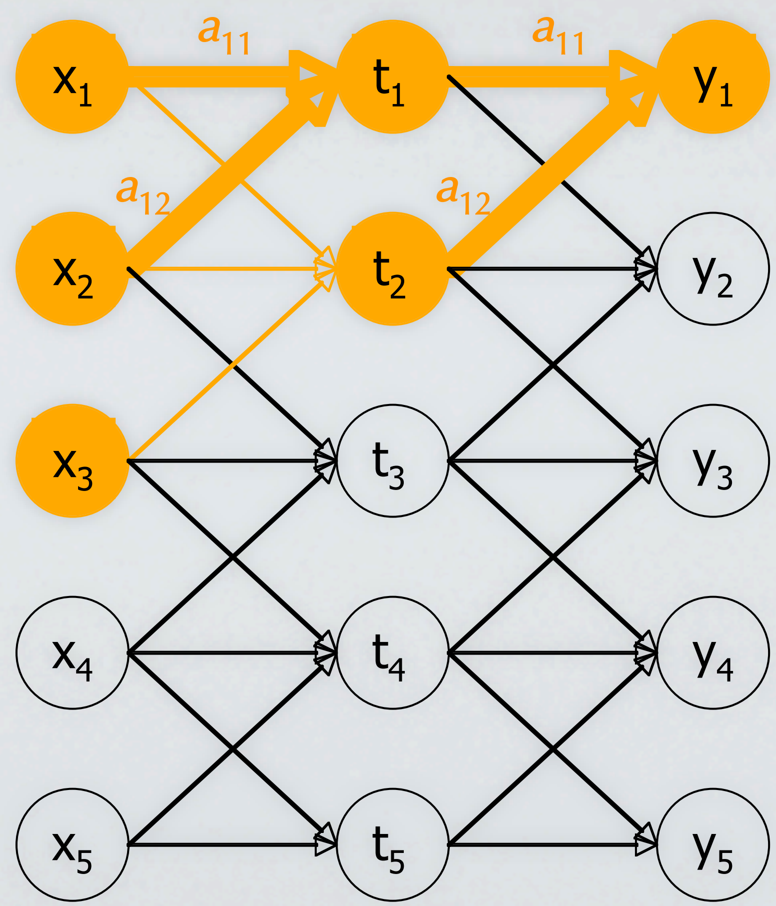
- I will post all materials at GT T-Square site for “CSE-8803-PNA”
  - Go to: http://t-square.gatech.edu
  - Note: Cross-listed with CS-8803-PNA / CS-4803-PNA, but ignore these
- Fill out survey questions under “Poll” tab
- When you “submit” the assignment, briefly describe what you hope to get out of this course
- Also: Use “Forum” to introduce yourselves

“In conclusion…”



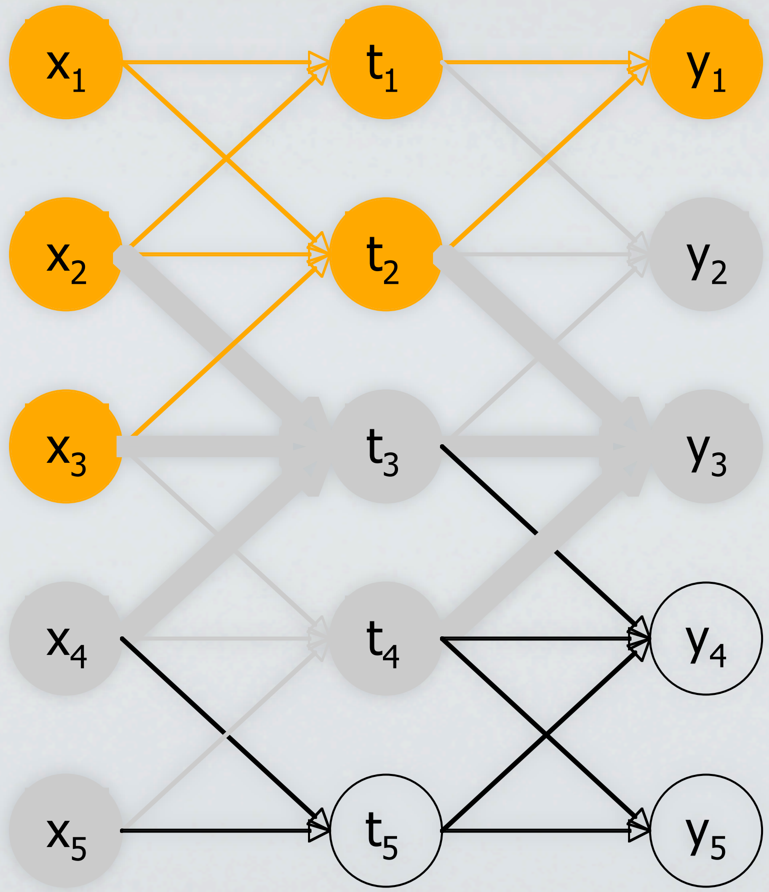
Need for locality and parallelism drives new algorithms.

Example: $y = A^2 * x$



Need for locality and parallelism drives new algorithms.

Example: $y = A^2 * x$



Need for locality and parallelism drives new algorithms.

Example: $y = A^{2*}x$. A new kernel implies a new algorithm… ?

Some questions raised by this example.

- This algorithm has locality, but what about parallelism?
  - What is the relationship between locality and parallelism?
- The “inner loop” of a solver based on this kernel is now A^{k}*x, not just A*x. How does this change the “outer loops?”

Technical challenges of PNA

- Precision, memory, and processing are finite resources
- Curse of dimensionality: 3D space + time eats up Moore's Law
- Hardware changes quickly
- Huge amount of *relevant* domain-knowledge exists: Collaborate!
- Parallel programming is hard