All About Erasure Codes:
- Reed-Solomon Coding
- LDPC Coding

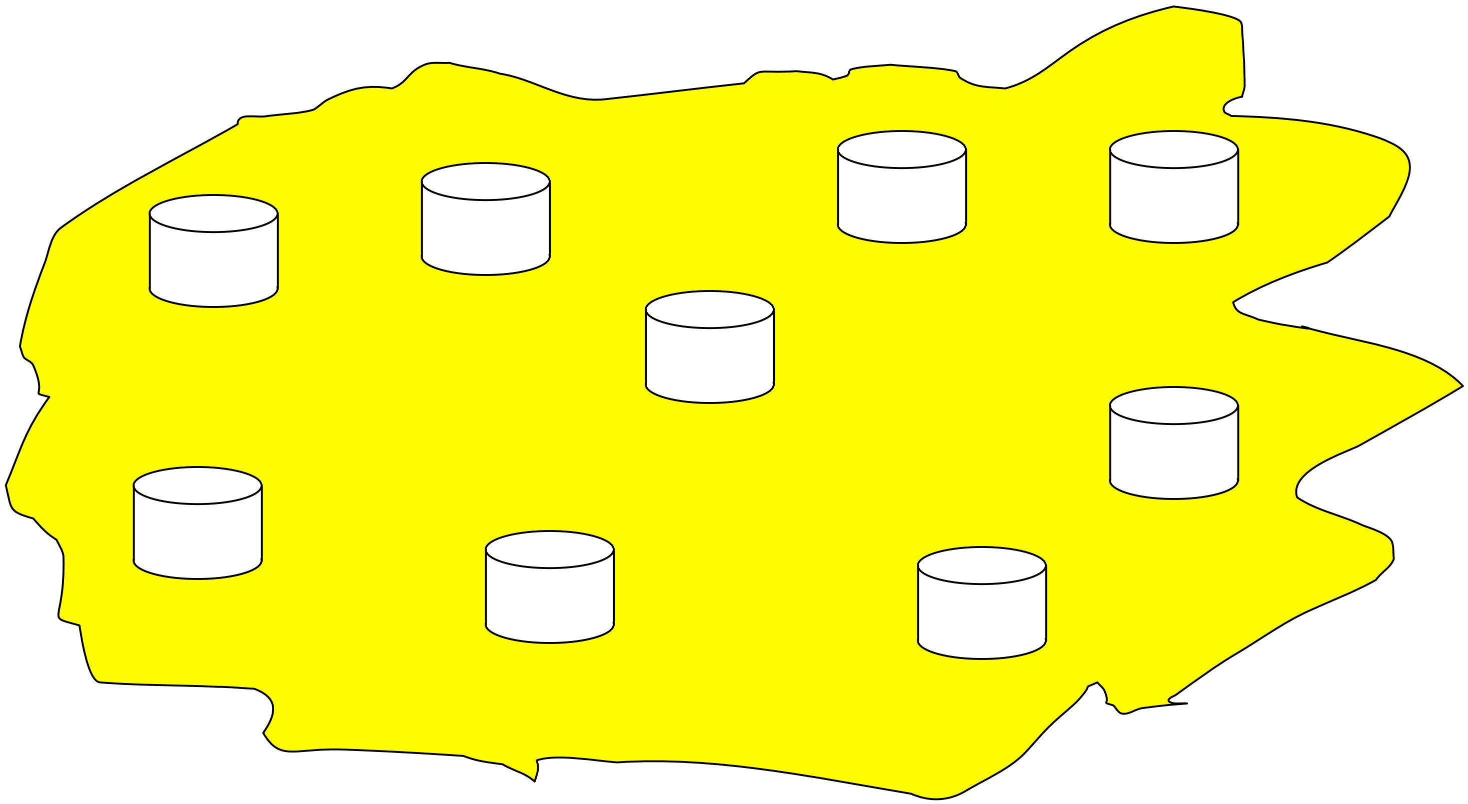
James S. Plank

Logistical Computing and Internetworking Laboratory
Department of Computer Science
University of Tennessee

ICL - August 20, 2004

Motivation

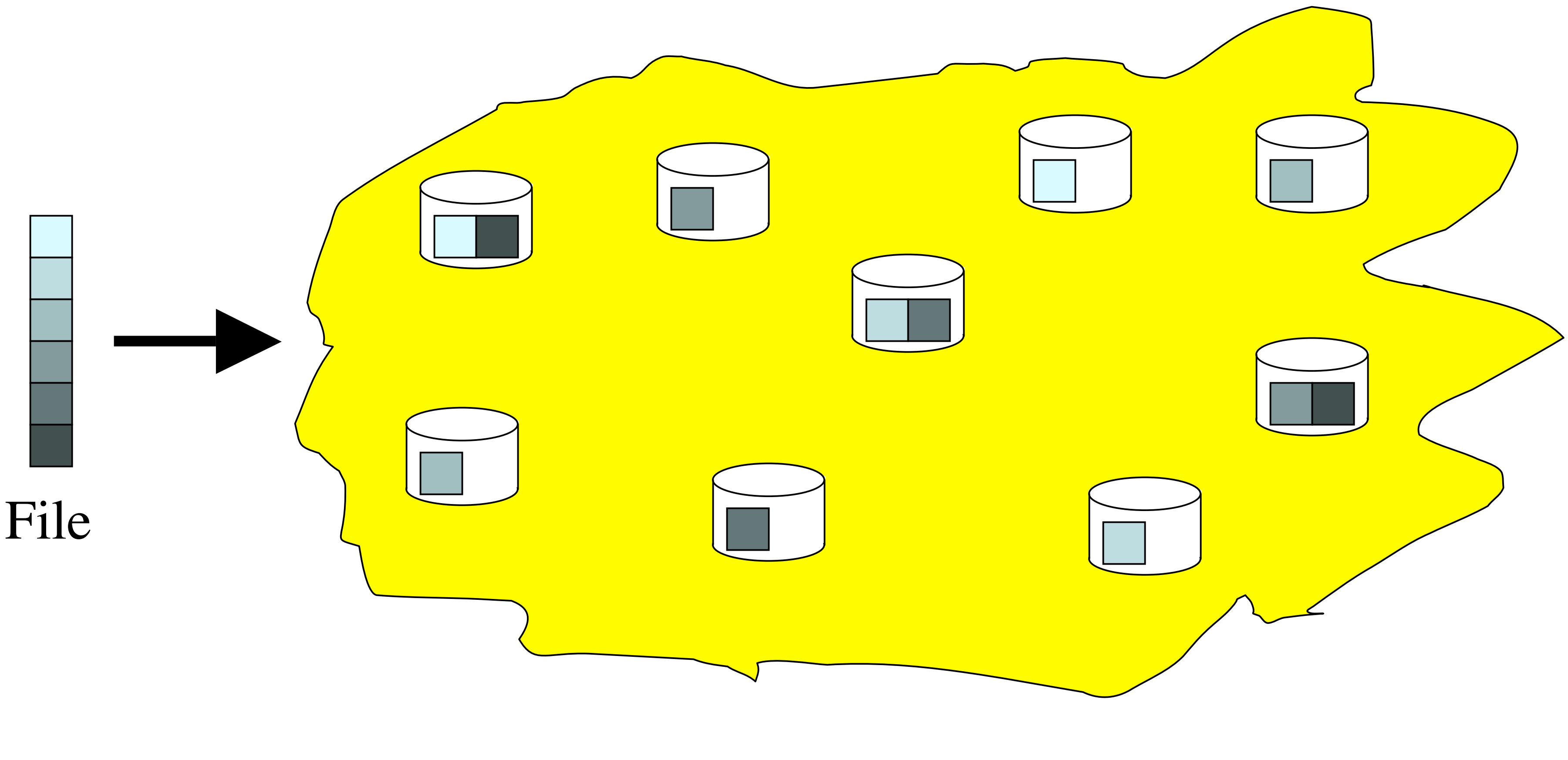
Behold a wide-area file system (grid, P2P, you name it):



Wide Area Network

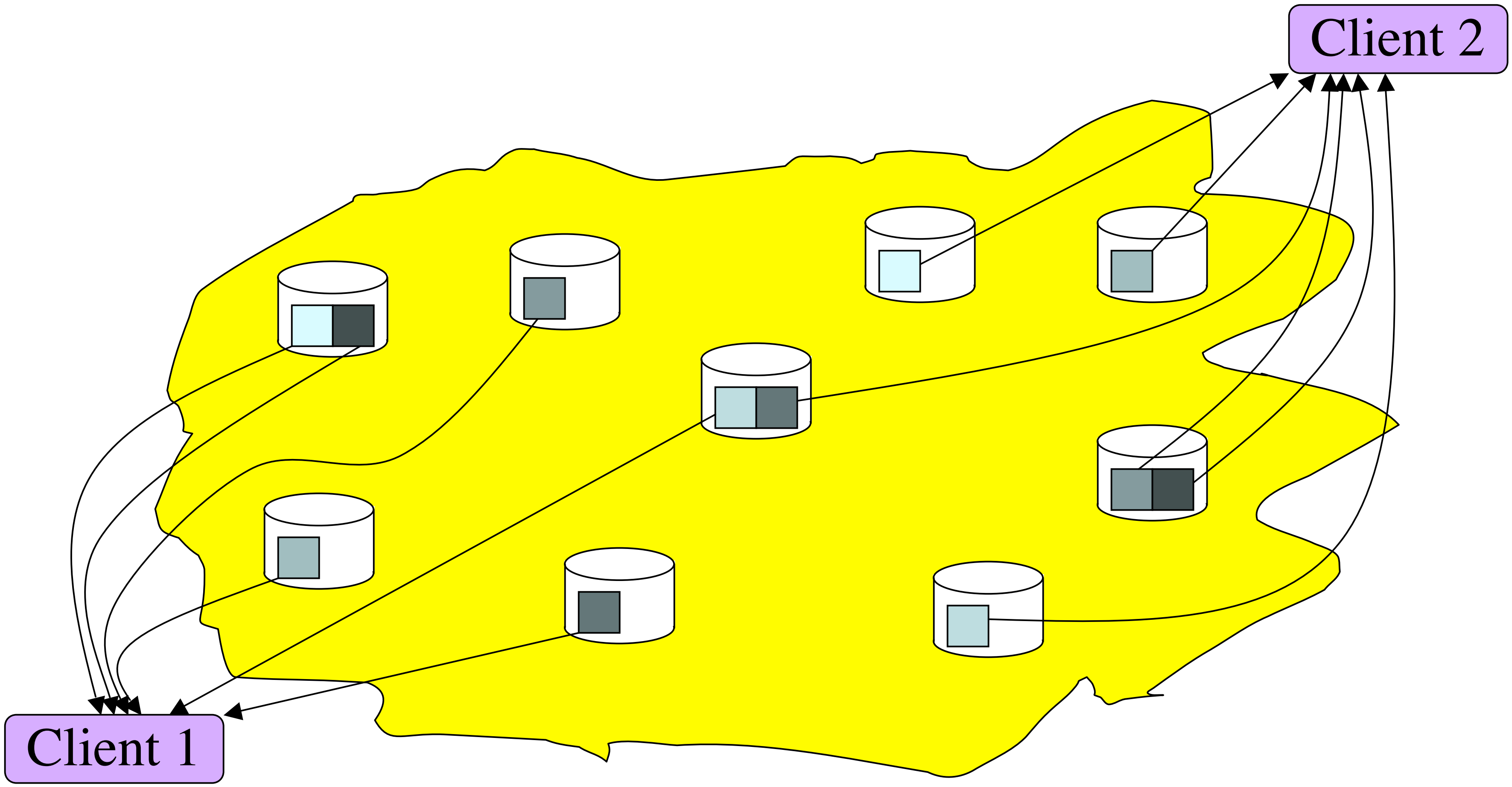
Motivation

Large files are typically partitioned into *n* blocks that are replicated among the servers:



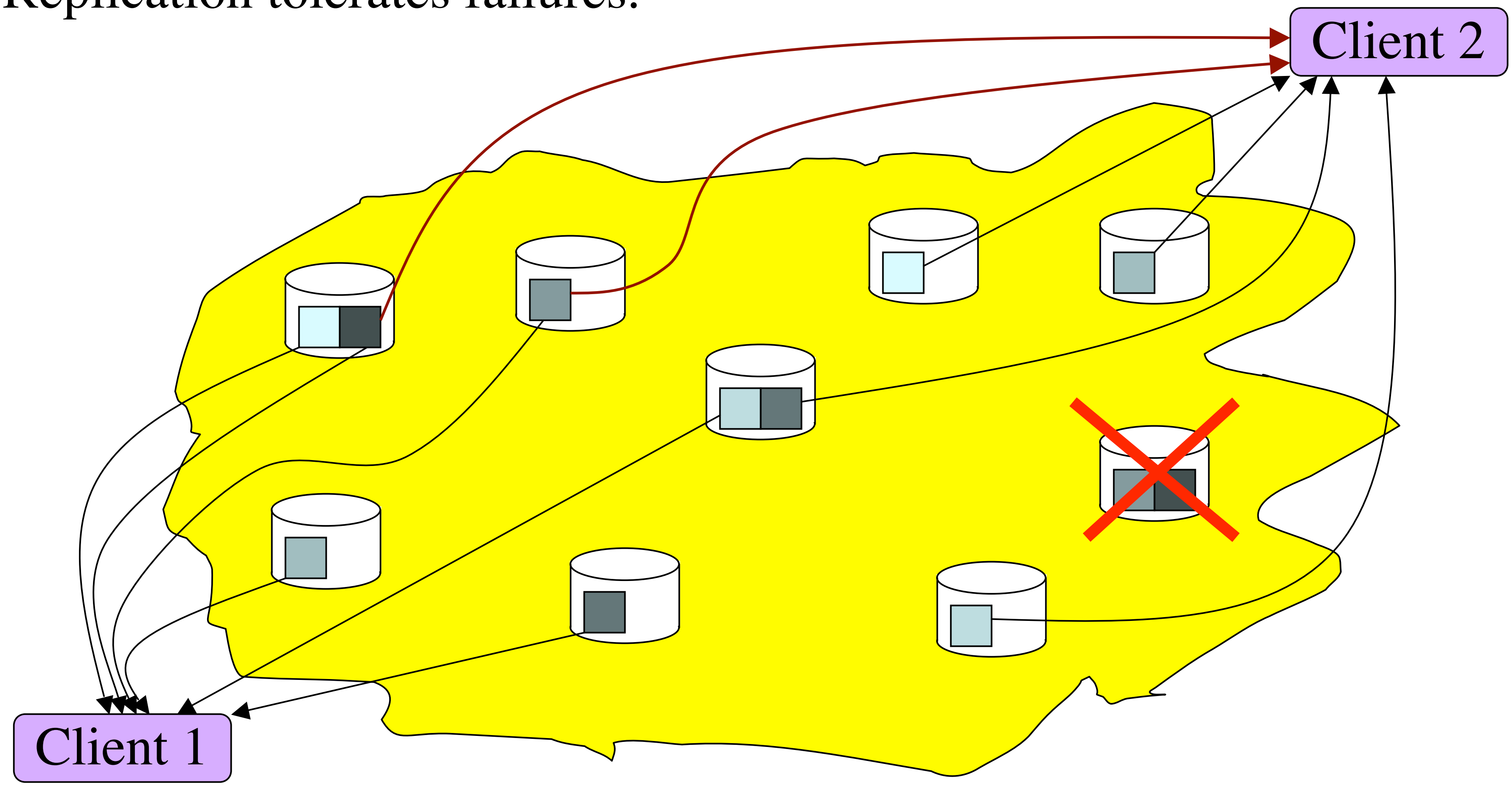
Motivation

Clients download the closest of each of the *n* blocks.



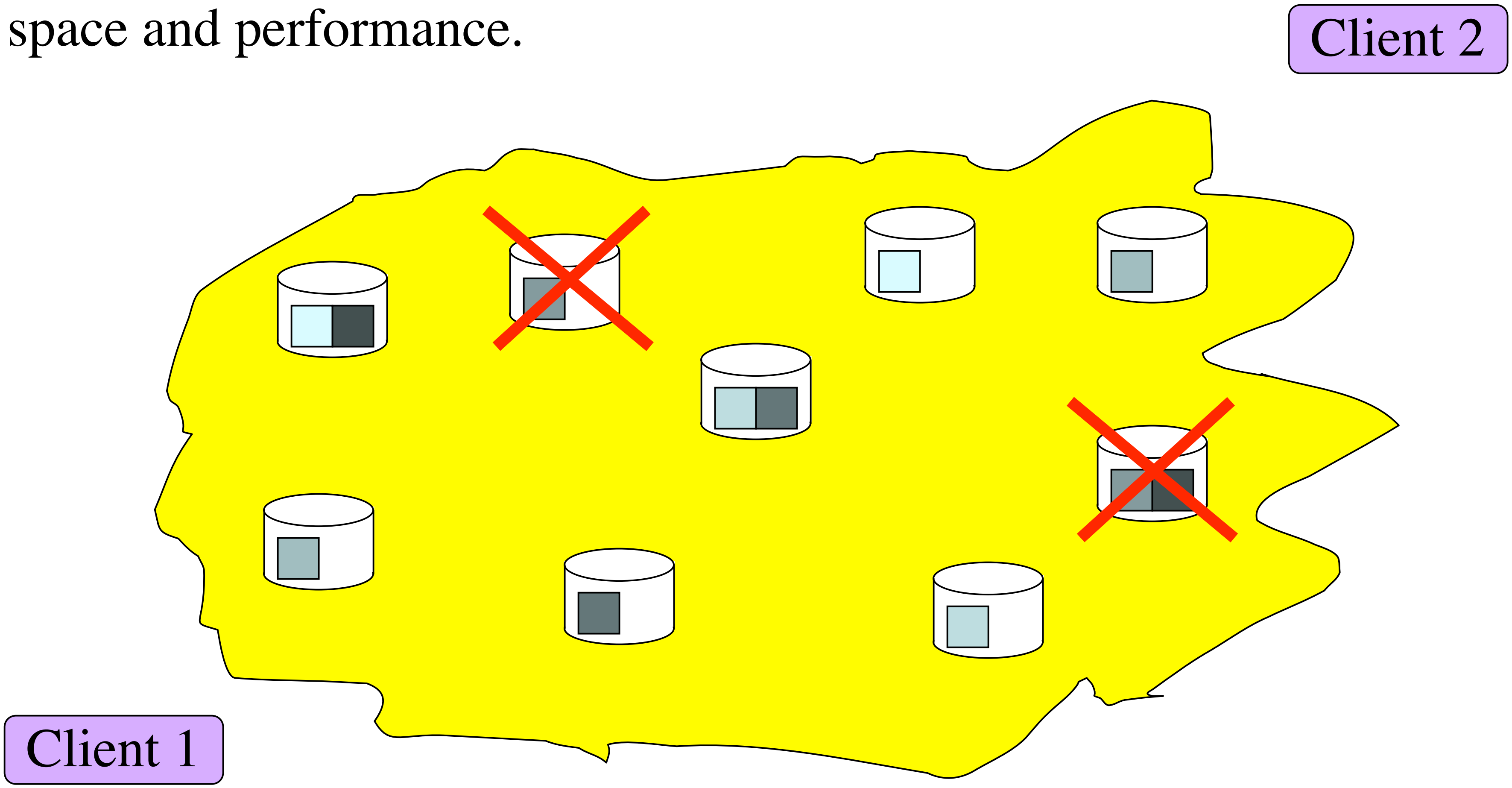
Motivation

Clients download the closest of each of the *n* blocks.
Replication tolerates failures.



Motivation

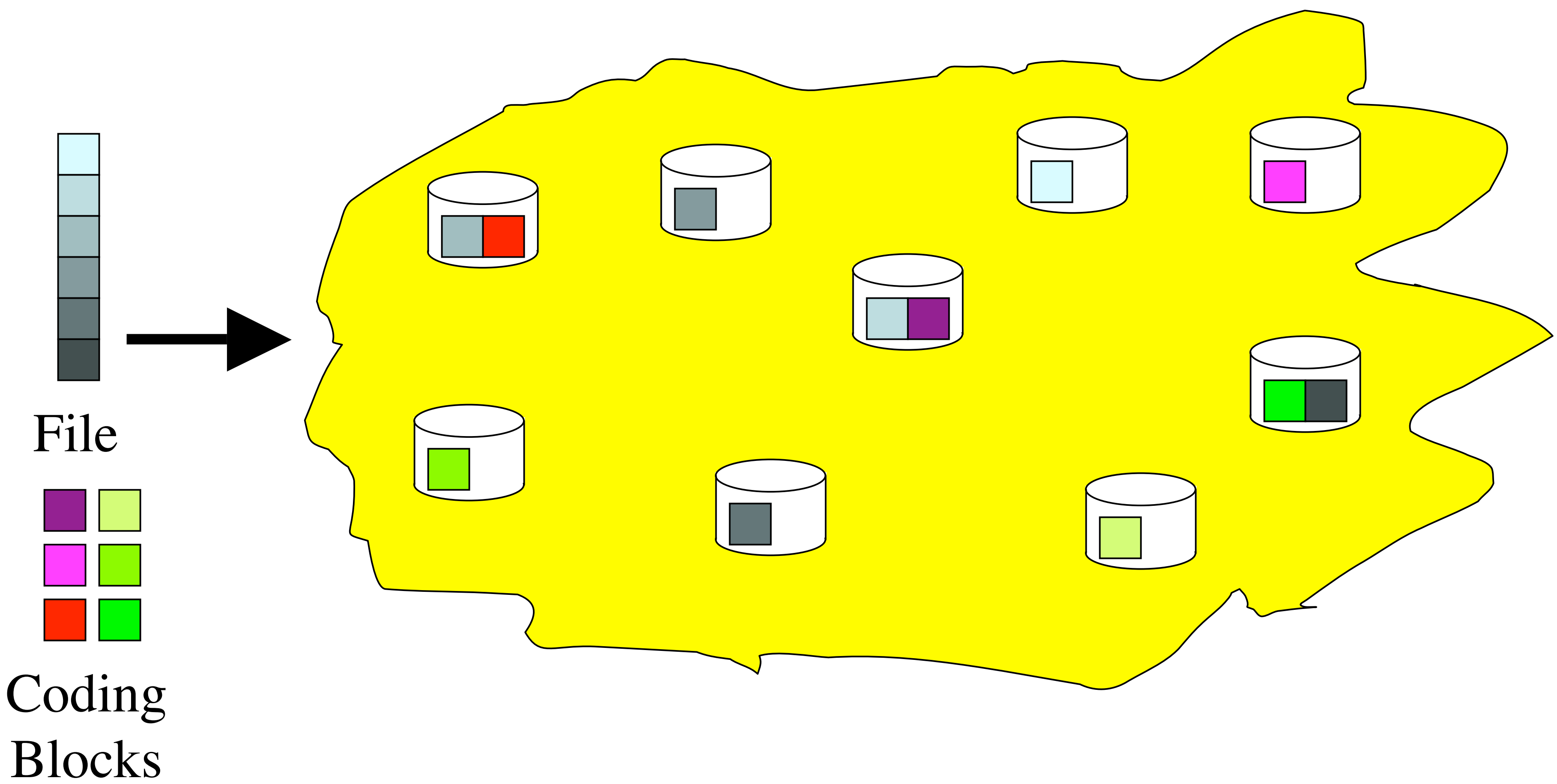
Unfortunately, replication is wasteful in terms of both space and performance.



Can’t get ▪, even though 9/12 blocks available.

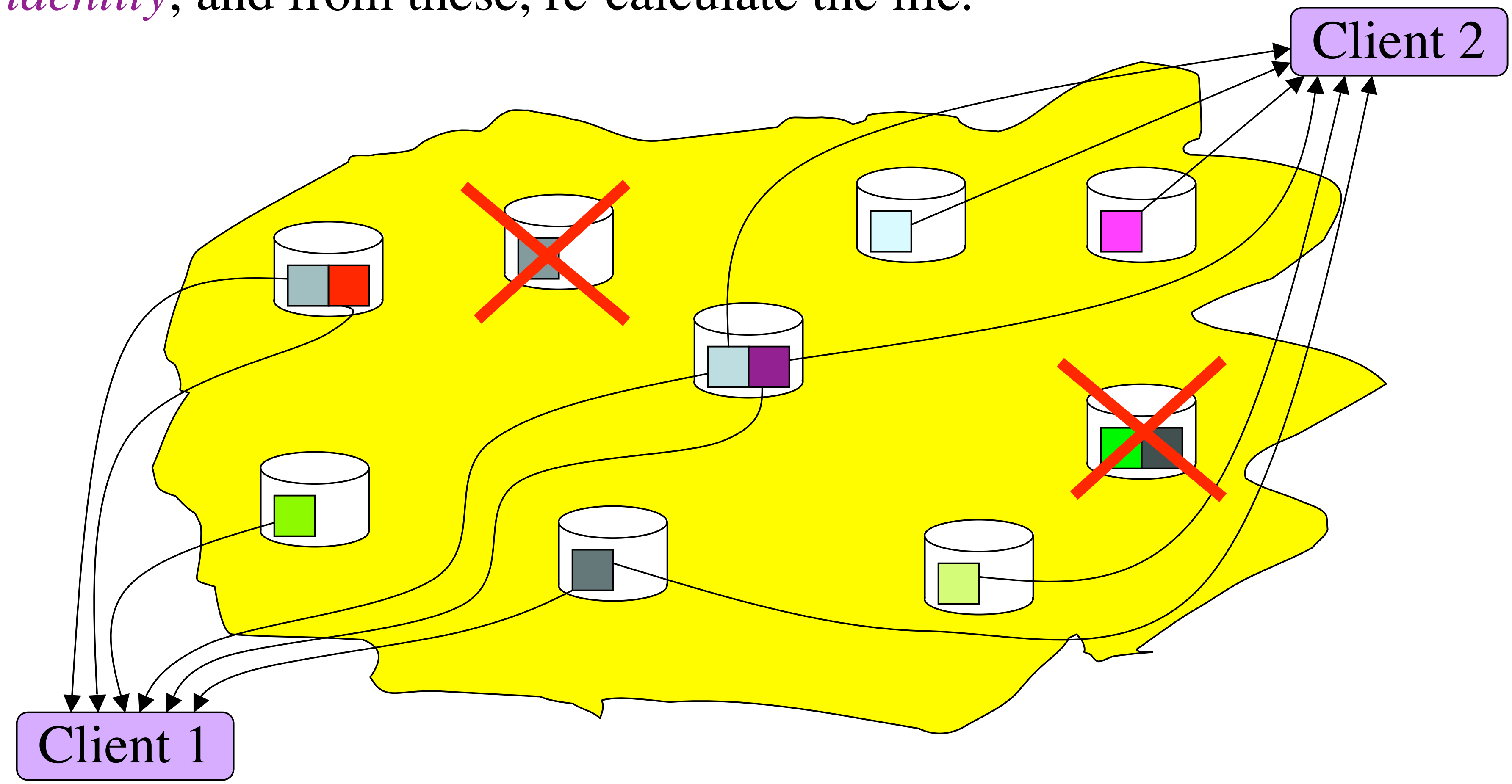
Motivation

Enter erasure codes -- calculate *m* coding blocks, and distribute the *n* + *m* blocks on the network.



Motivation

Clients download the *n*+□ closest blocks, *regardless of identity*, and from these, re-calculate the file.



This is a good thing

- *Excellent space used / fault-tolerance.*
- *Relief from block identity* -- any n+□ blocks will do.
- *However*:
  - Historical codes (Reed-Solomon) have performance issues.
  - More recent codes have patent issues.
  - More recent codes are open research questions.

- ***Bottom Line****: Realizing the promise of erasure coding is not a straightforward task.*

The Outline of This Talk

- *Primer on Reed-Solomon Codes*
- *History of LDPC Codes*
- *Practical Evaluation of LDPC Codes*
- *Optimal, Small LDPC Codes*
- *Reed-Solomon Codes in LoRS*

Primer on Reed-Solomon Codes

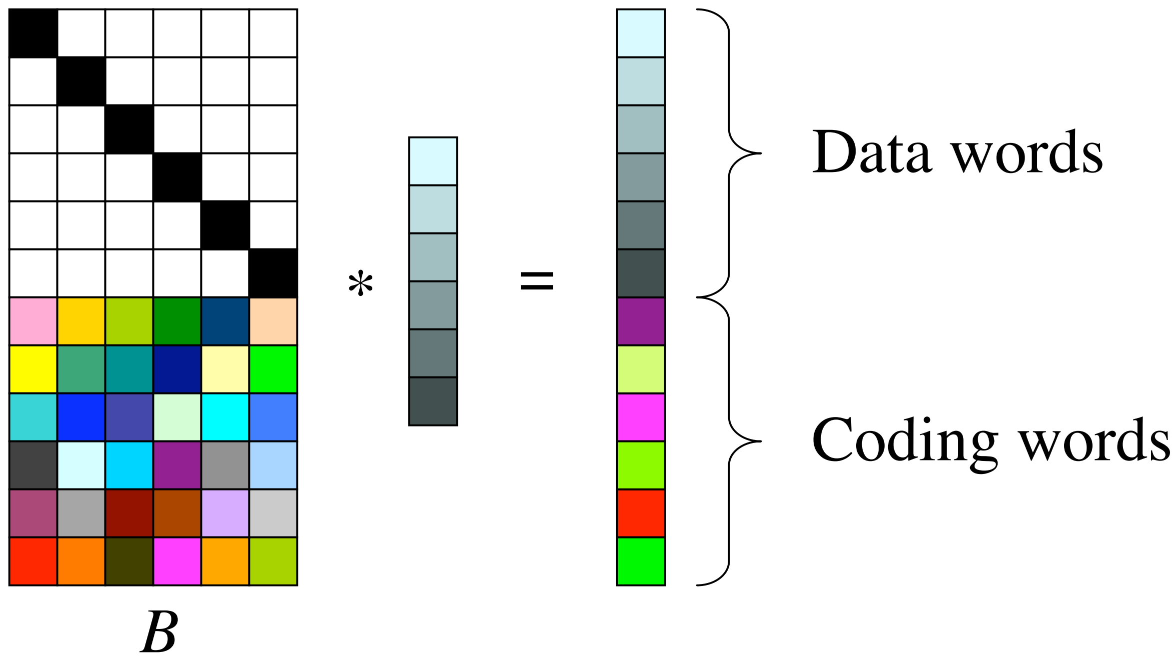
Reed-Solomon Coding is the canonical erasure code:

- Suppose we have n data devices & m coding devices
- Break up each data block into words of size $w \mid 2^w < n+m$
- There are n data words $d_1, \ldots, d_n$
- And m coding words $c_1, \ldots, c_m$
- Encoding & decoding revolve around an $(n+m)$ X n coding matrix B.

Primer on Reed-Solomon Codes

- Define an *(n+m)* X *n* coding matrix *B* such that:

$$B <d_1, \ldots, d_n> = <d_1, \ldots, d_n, c_1, \ldots, c_m>$$

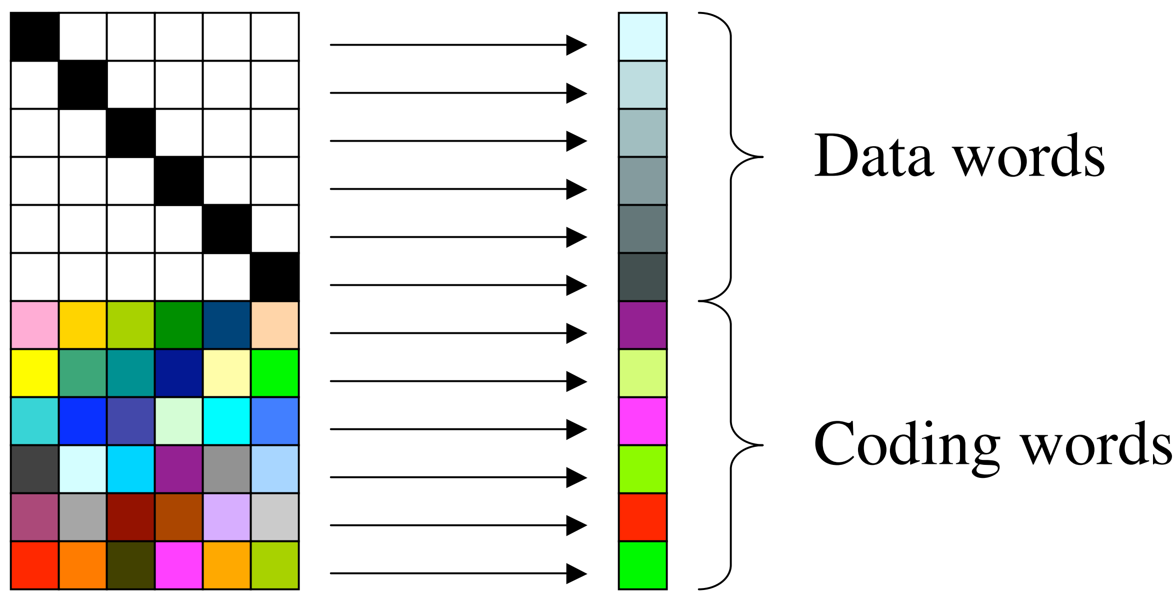


- *B* must have an additional property that all *n* X *n* matrices created by deleting *m* rows from *B* are invertible.

Primer on Reed-Solomon Codes

B derived from "Vandermonde" matrix; Guaranteed to exist.

To decode, first note that every row of *B* corresponds to a data or coding word.



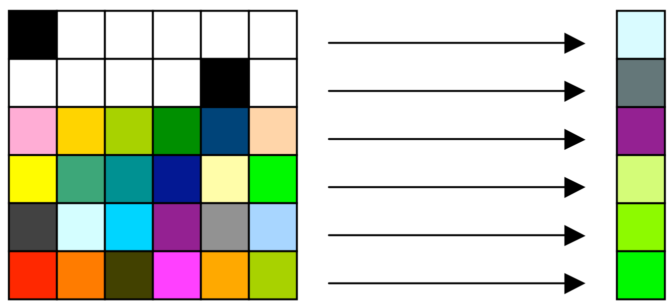
B

Primer on Reed-Solomon Codes

Decoding:

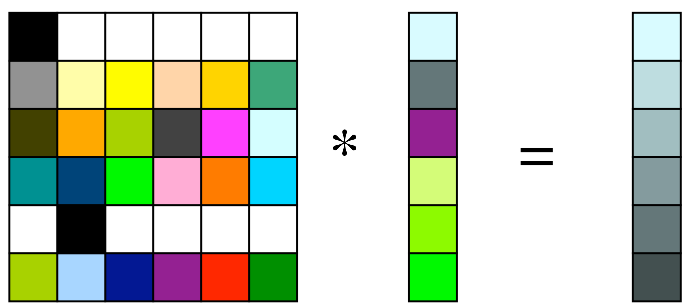
Suppose you download n words. Create B' from the n rows of B corresponding to those n words.

B'



Now, invert B':

B'^{-1} * existing words = data words



B'^{-1}

RS-Coding Details

- Must use *Galois Field* arithmetic $GF(2^w)$
  - Addition = exclusive or: *cheap*
  - Multiplication/Division requires log & anti-log lookup tables: *more expensive*

- Encoding is $O(mn)$.
- Decoding requires:
  - n X n matrix inversion: $O(n^3)$,
  - Then $O(n^2)$ to recalculate data words.

- However, with x words per block:
  - Encoding is $O(mnx)$.
  - Decoding is $O(n^3) + O(n^2x)$.

- Bottom line: When n & m grow, it is brutally expensive.

A Watershed in Coding Theory

In 1997, Luby *et al* introduced the world to:
Tornado Codes:

- *Good Properties*:
  - Calculations involve parity (XOR) only.
  - Each block requires a fraction of the other blocks to calculate -- Encoding & Decoding: $O(x(n+m))$.

- *Bad Properties*:
  - □ > 0 -- I.e. you need > n blocks to recalculate the file.
  - Theory developed for asymptotics & not well understood in the finite case.

History

- *The Luby 1997 paper is a landmark*:
  - In 1998, Byers *et al* show how Tornado Codes can greatly outperform Reed-Solomon codes for large values of n.
  - Luby *et al* soon form Digital Fountain, and patent their codes.
  - Scores of people publish studies on similar “LDPC” codes with asymptotically optimal properties.
- *However…*

History

No one studies the practical implications of these codes!!!!

Which Means:

They remain unusable for developers of wide-area storage systems!!!!!!!

- Why?
  - *Hard-core graph theory scares off systems people.*
  - *Hard-core graph theorists like asymptotics & theory...*
  - *Patent worries scare off potential implementers.*
- The Bottom Line
  - *There is nowhere to find a "Tornado Code" for your storage system.*
  - *Therefore, we (LoCI, OceanStore, BitTorrent) use Reed-Solomon codes.*

The Mission of Our Research:

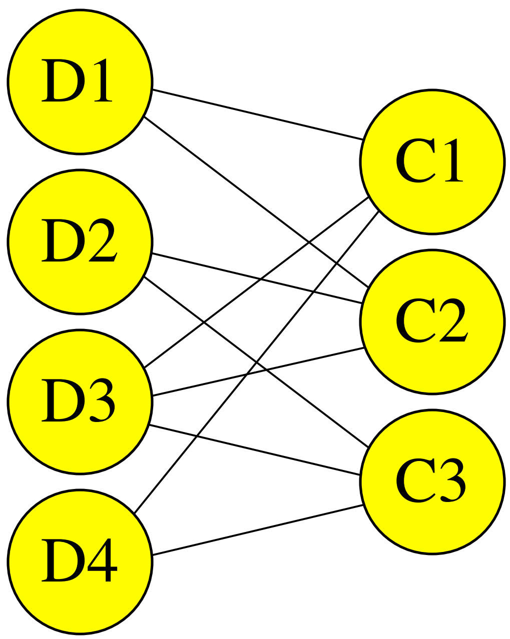
To study the practical properties of LDPC codes for Wide-Area Storage Systems

- To *quantify* their performance in wide-area systems.
- To *explore* various facets of code generation.
- To *compare* their performance to Reed-Solomon coding.
- To *raise* important research questions for the theoretical community.

LDPC Codes

Low-Density, Parity-Check Codes

- Simplest incarnations are codes based on bipartite graphs -- data bits on the left, coding on the right.



C1 = D1+D3+D4

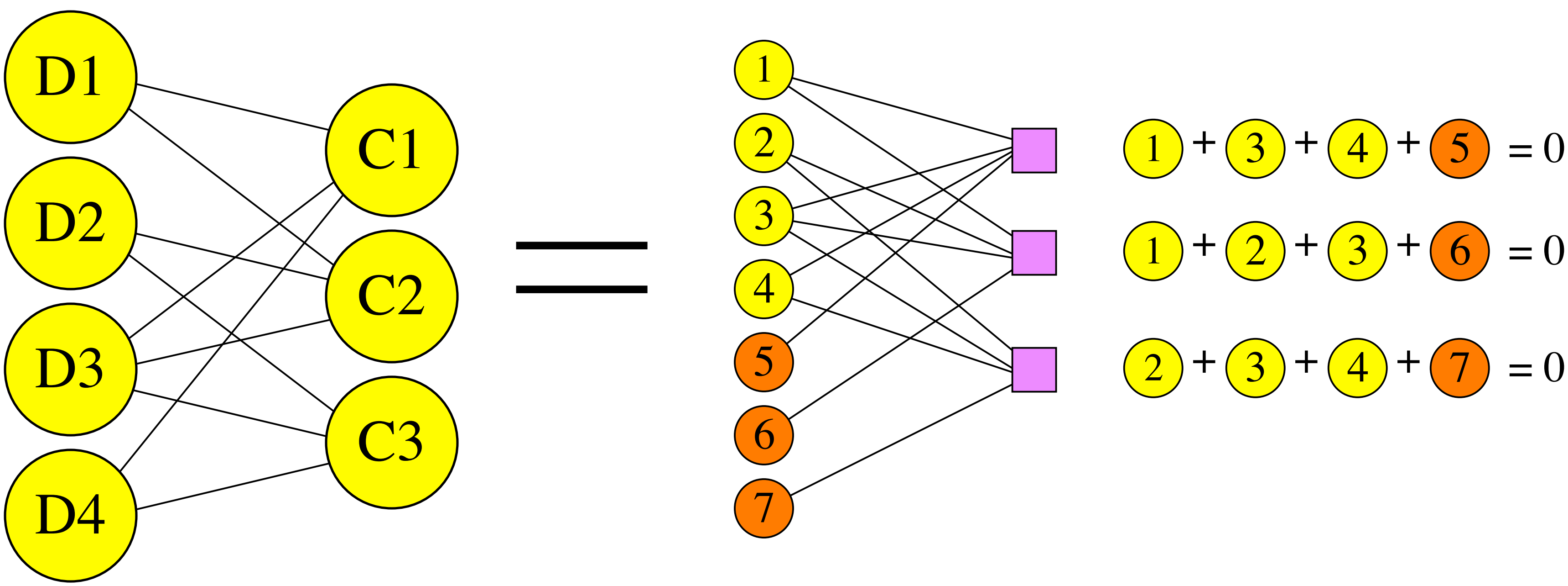
C2 = D1+D2+D3

C3 = D2+D3+D4

LDPC Codes

Tanner Graph representation

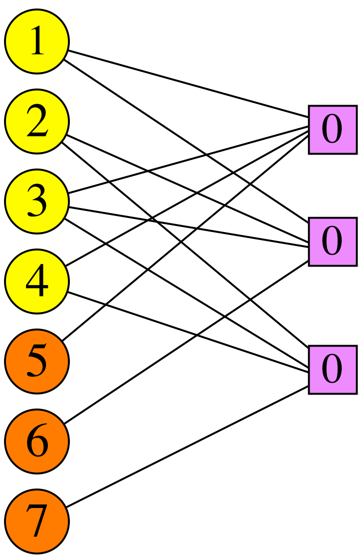
- Alternative representation -- all data and coding bits on the left, and *constraints* on the right:



Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes & XOR downloaded/calculated values into constraints.

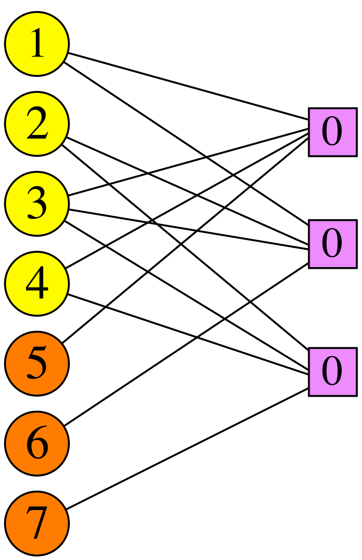
Start with 0's in constraints



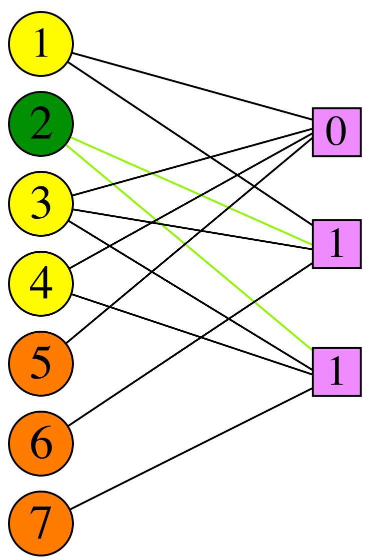
Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes & XOR downloaded/calculated values into constraints.

Start with 0's in constraints



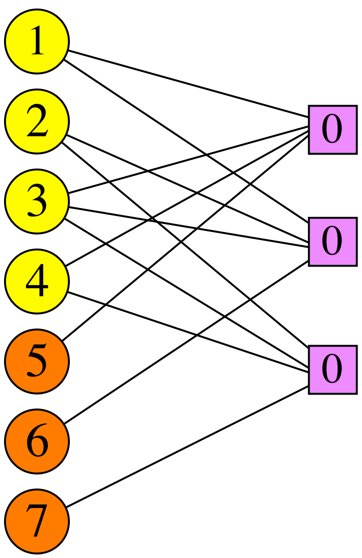
Download bit 2 as 1:



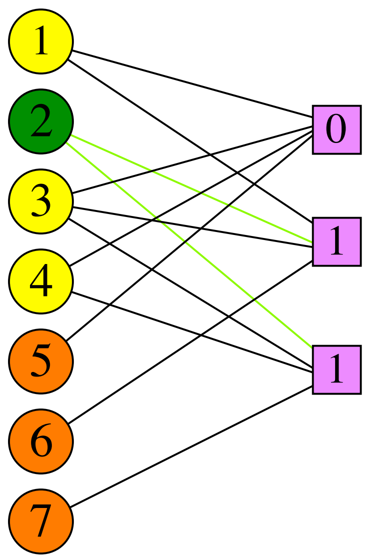
Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes & XOR downloaded/calculated values into constraints.

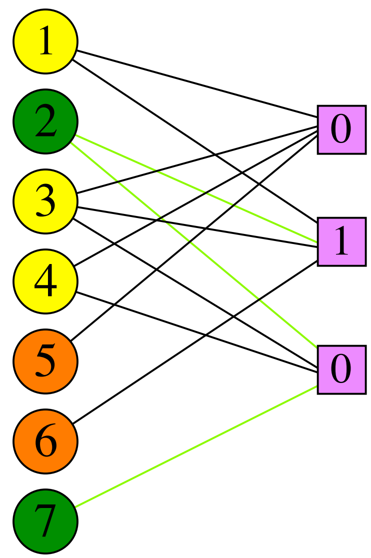
Start with 0's in constraints



Download bit 2 as 1:



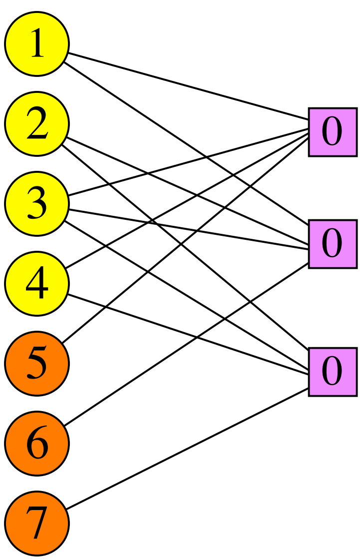
Download bit 7 as 1:



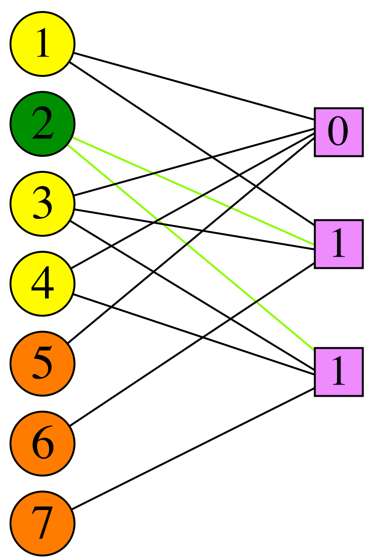
Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes & XOR downloaded/calculated values into constraints.

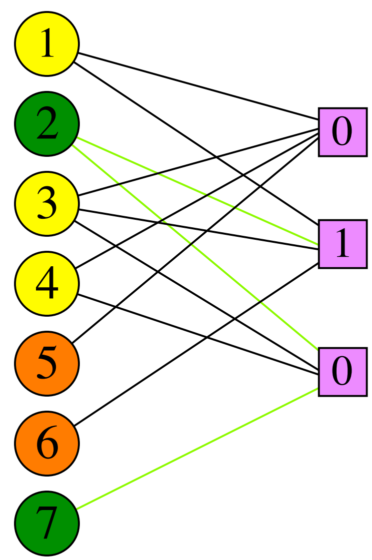
Start with 0's in constraints



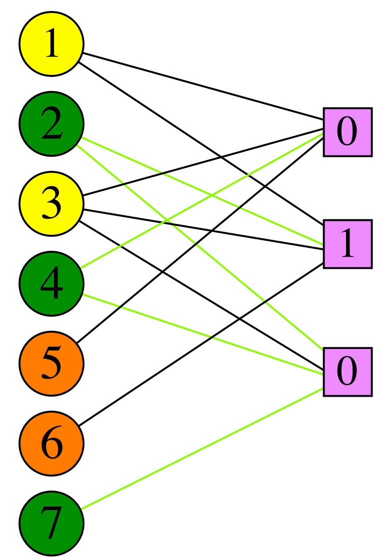
Download bit 2 as 1:



Download bit 7 as 1:



Download bit 4 as 0:

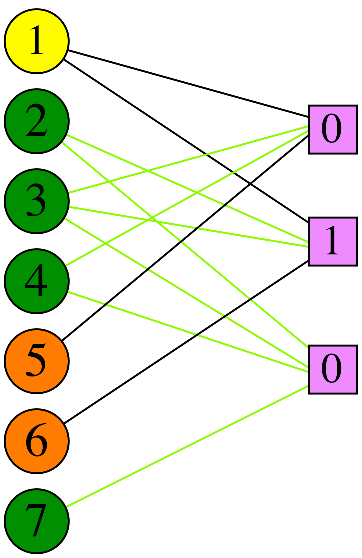


Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes.

If a constraint only has one edge,
the constraint holds the connected node's value:

Determine
bit 3 as 0,
from constraint 3:

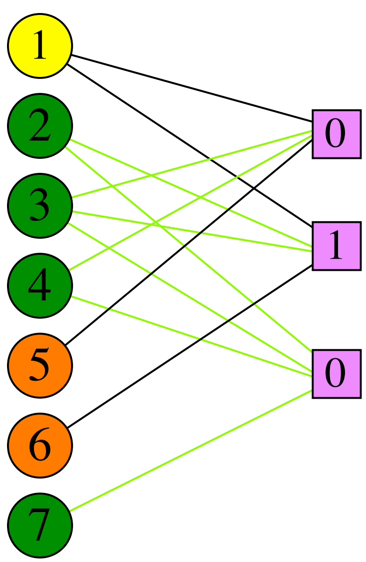


Two good things about Tanner Graphs

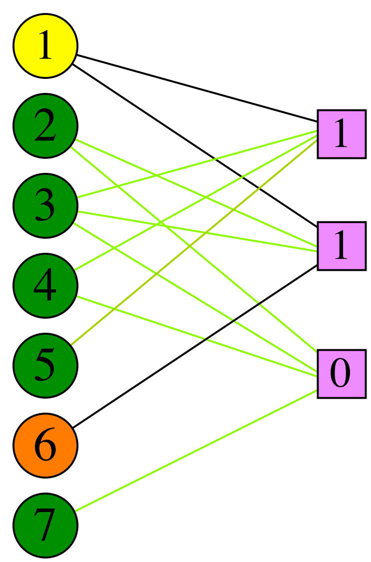
- #1: Decoding easy to define: Just remove nodes.

If a constraint only has one edge,
the constraint holds the connected node's value:

Determine
bit 3 as 0,
from constraint 3:



Download
bit 5 as 1:

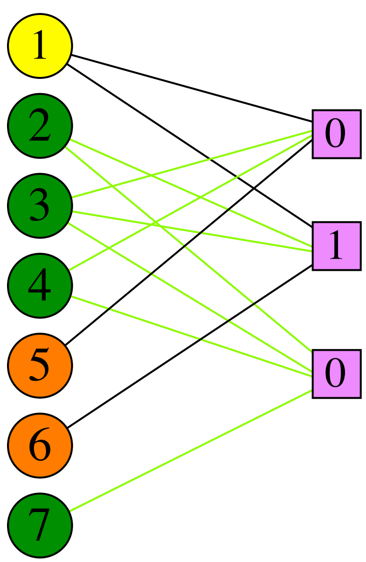


Two good things about Tanner Graphs

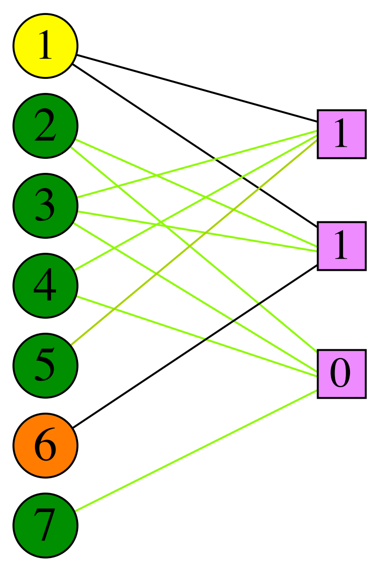
- #1: Decoding easy to define: Just remove nodes.

If a constraint only has one edge,
the constraint holds the connected node's value:

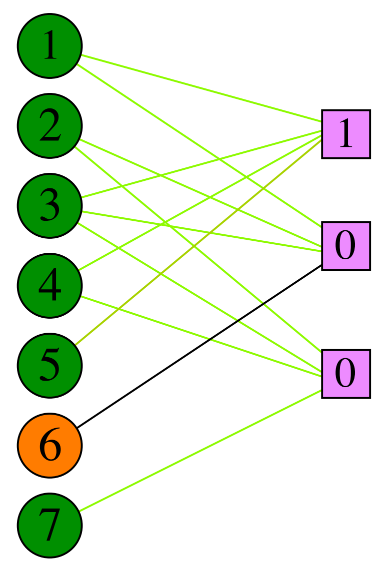
Determine
bit 3 as 0,
from constraint 3:



Download
bit 5 as 1:



Determine
bit 1 as 1,
from constraint 1:

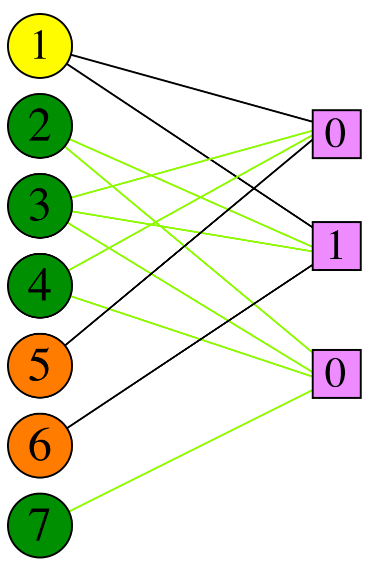


Two good things about Tanner Graphs

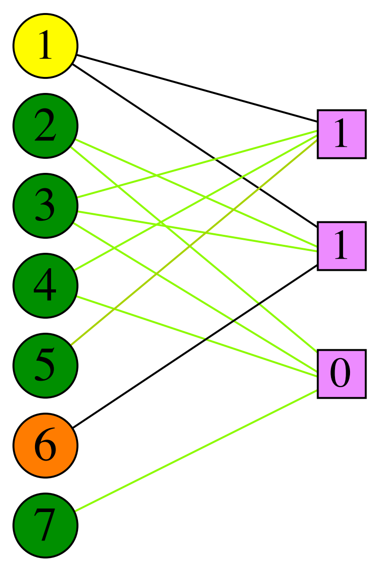
- #1: Decoding easy to define: Just remove nodes.

If a constraint only has one edge,
the constraint holds the connected node's value:

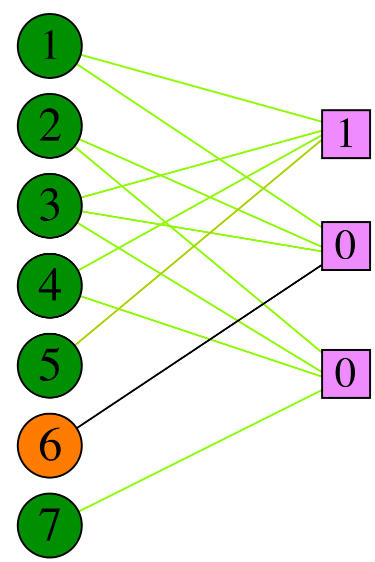
Determine
bit 3 as 0,
from constraint 3:



Download
bit 5 as 1:



Determine
bit 1 as 1,
from constraint 1:



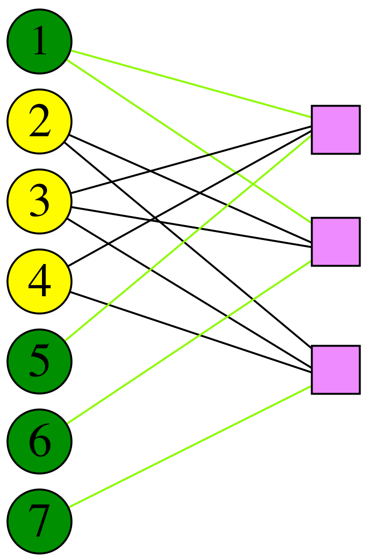
And finally
bit 6 is 0.

Two good things about Tanner Graphs

- #1: Decoding easy to define: Just remove nodes.

Note -- it may take > n downloaded bits to decode:

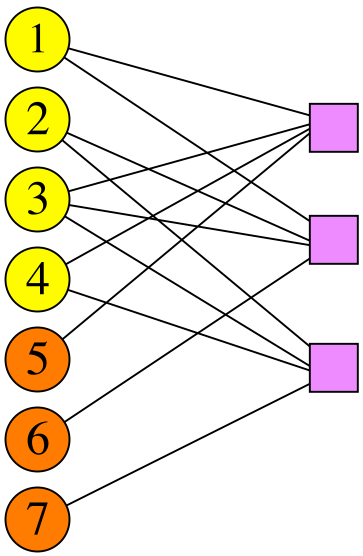
Suppose we download bits 1 5 6 & 7:



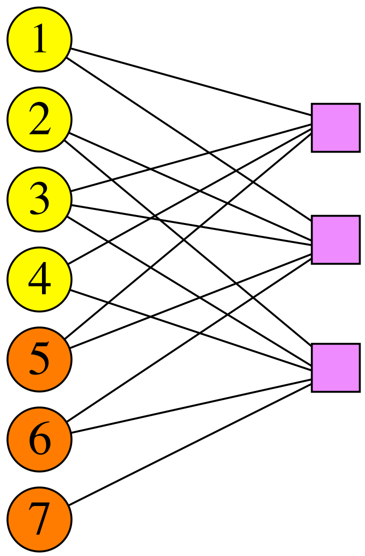
-- We still cannot determine the undownloaded bits.

Two good things about Tanner Graphs

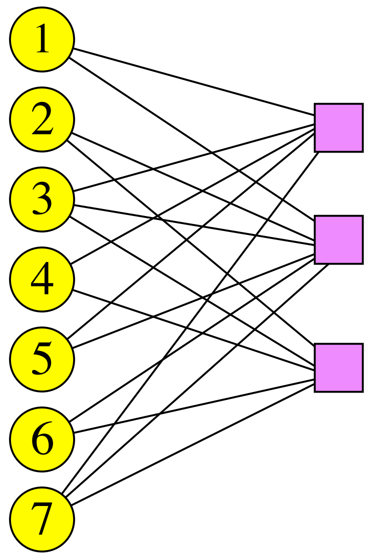
- #2: Can represent more complex codes.



Simple Systematic



More Complex Systematic



Non-Systematic

"Systematic" = data bits are part of the left-hand nodes.

Classes of LDPC Codes

- *Gallager codes* -- first developed in the 1960's, but received further attention since Luby's 1997 paper. Encompasses all codes represented with Tanner graphs -- non-systematic codes require matrix operations for encoding/decoding.

- *Simple Systematic Codes* -- Systematic codes where each coding node has just one edge to a unique constraint.

- *Tornado codes* -- Simple systematic codes that cascade.

- *IRA codes* -- Systematic codes, where coding each coding node i has edges to constraints i and $(i+1)$.

Similarities of these codes

- All based on *bipartite graphs*.
- Graphs define *parity operations* for encoding/decoding.
- Decoding overhead based on # of edges in graph *(low density).*
- All have been proven to be *asymptotically optimal*.

History: Nature of Theory

- Choose a rate $R = n/(n+m)$ for the code.
- Define *probability distributions* □ and □ for cardinality of left-hand and right-hand nodes.
- Define f to be the *overhead factor* of a graph:
  - On average, fn nodes of the $(n+m)$ total nodes must be downloaded to reconstitute the data.
  - $f = 1$ is optimal (like Reed-Solomon coding).
- Prove that for *infinite graphs* where node cardinalities adhere to □ and □, f is equal to one.
- QED.

Questions We Strive To Answer

1. What kind of overhead factors *(f)* can we expect for LDPC codes for large and small *n*?
2. Are the three types of codes equivalent or do they perform differently?
3. How do the published distributions fare in producing good codes for finite values of *n*?
4. Is there a great deal of random variation in code generation for given probability distributions?
5. How do the codes compare to Reed-Solomon coding?

Experimental Methodology

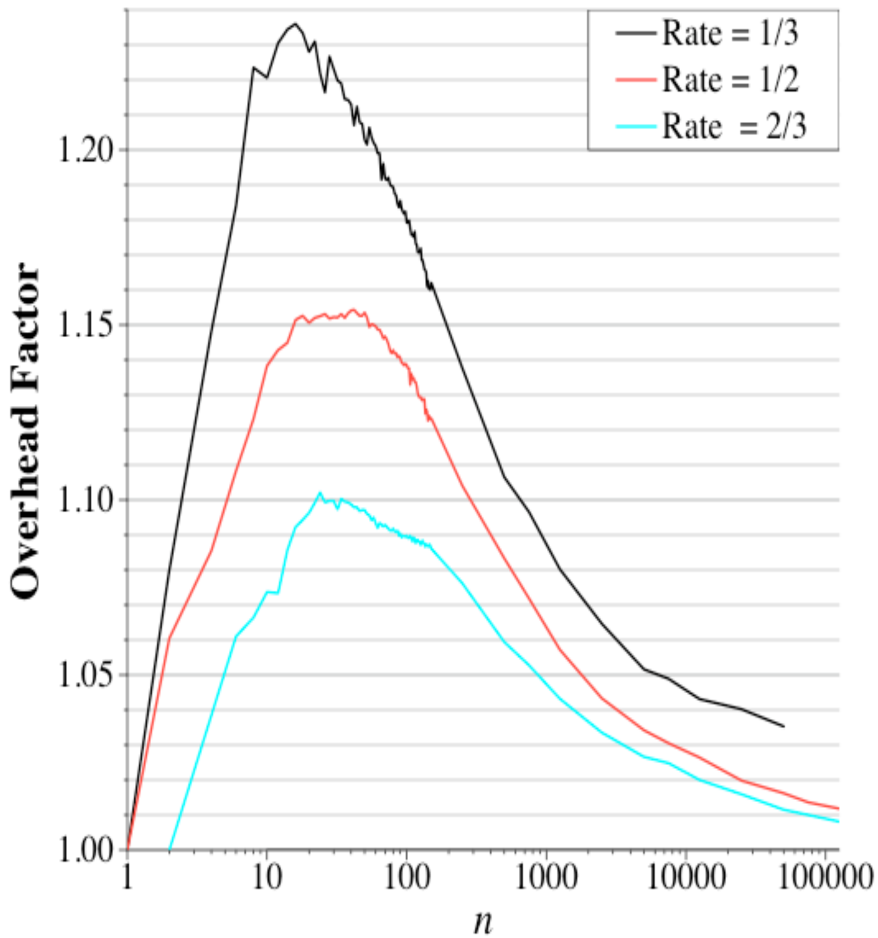
- Choose R.
- Choose n.
- Calculate m from $R = n/(n+m)$.
- Generate a graph in one of three ways:
  - Use a published probability distribution.
  - Use a probability distribution derived from a previously generated graph.
  - Use a randomly generated probability distribution.
- Perform a Monte-Carlo simulation of 1000's of random downloads, and experimentally determine the average f.

Data Points

- $R \in \{1/2, 1/3, 2/3\}$.
- Small $n \in$ {Even numbers between 2 and 150}.
- Large $n \in$ {250, 500, 1250, 2500, 5000, 12500, 25000, 50000, 125000}.
- 80 published probability distributions for all graphs and rates.
- Derive from "nearby" best graphs.

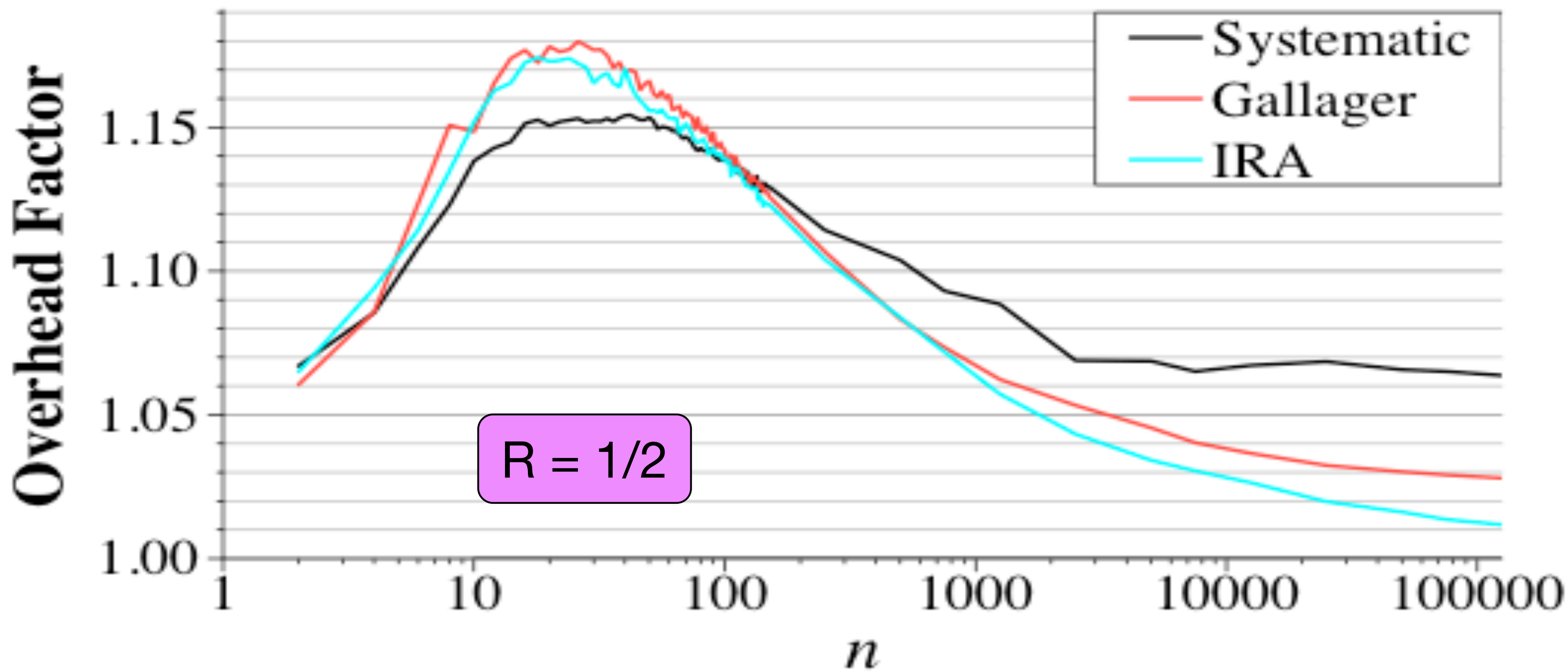
Total: Over 200,000 data points, each repeated over 100 times with different seeds.

Q1: Best Overhead Factors



- All rise to a maximum with 10 < n < 50, then descend toward 1 as n approaches .
- Larger rates perform better.
- Open Questions:
  - Upper bounds for given n?
  - Lower bounds for given n?
  - Can the shape of the curves be defined precisely ?

Q2: Three Codes Same or Different?



- They're different.
- Systematic best for small *n*.
- IRA best for large *n*.
- Other rates similar.

- Open questions:
  - What gives? Why are we seeing what we're seeing?
  - How can Systematic or IRA outperform Gallager?

Q3: How Do Published Codes Fare?

- W.R.T. small n: very poorly.

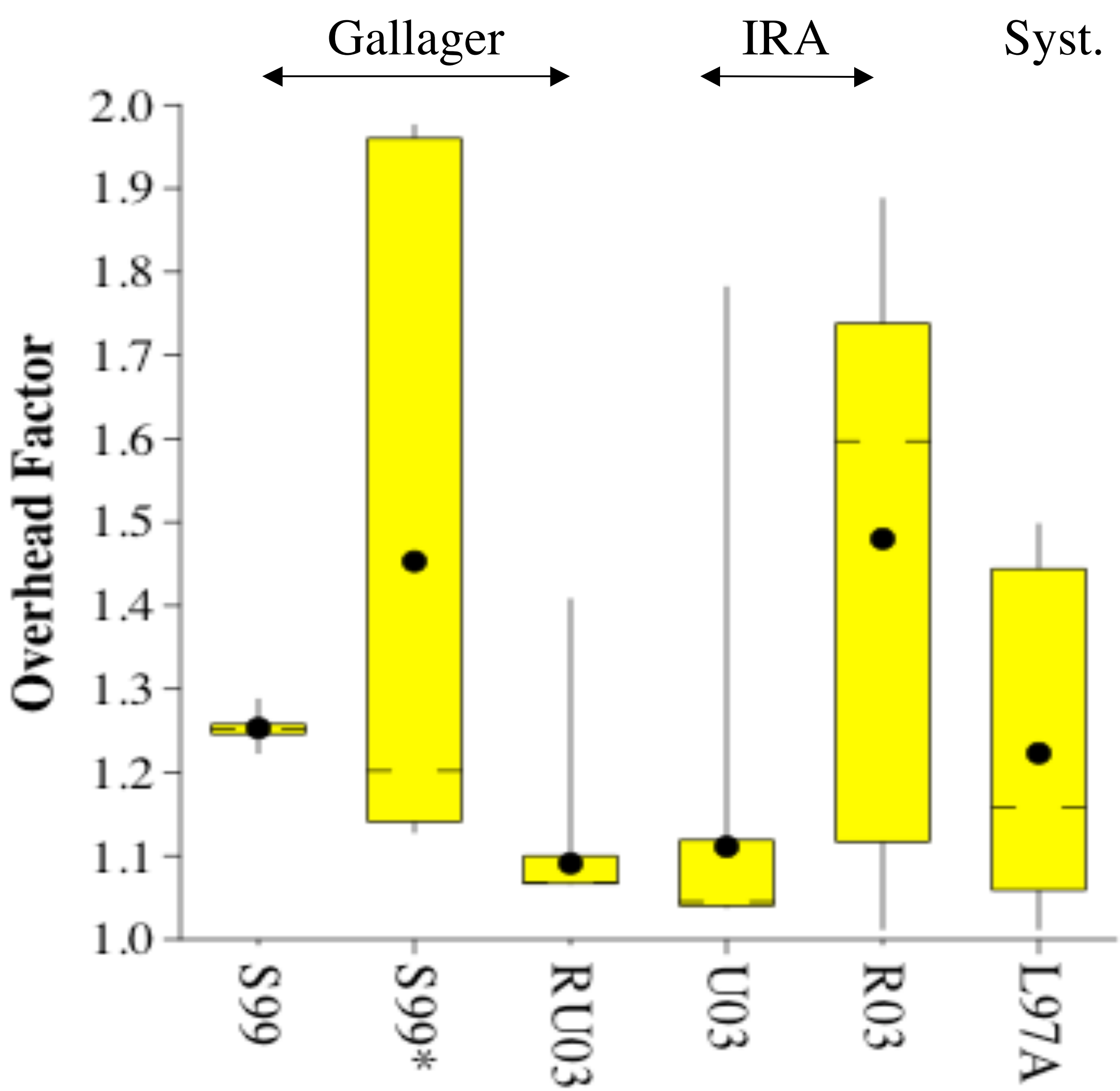
- W.R.T. large n: very poorly, except in certain cases.

- <u>Open Questions</u>:

  – Although the codes converge to f=1 as n goes to   , parameterizing □ and □ to minimize f for small n is clearly an open question.

  – What about other rates?

Q4: Variation in Performance?

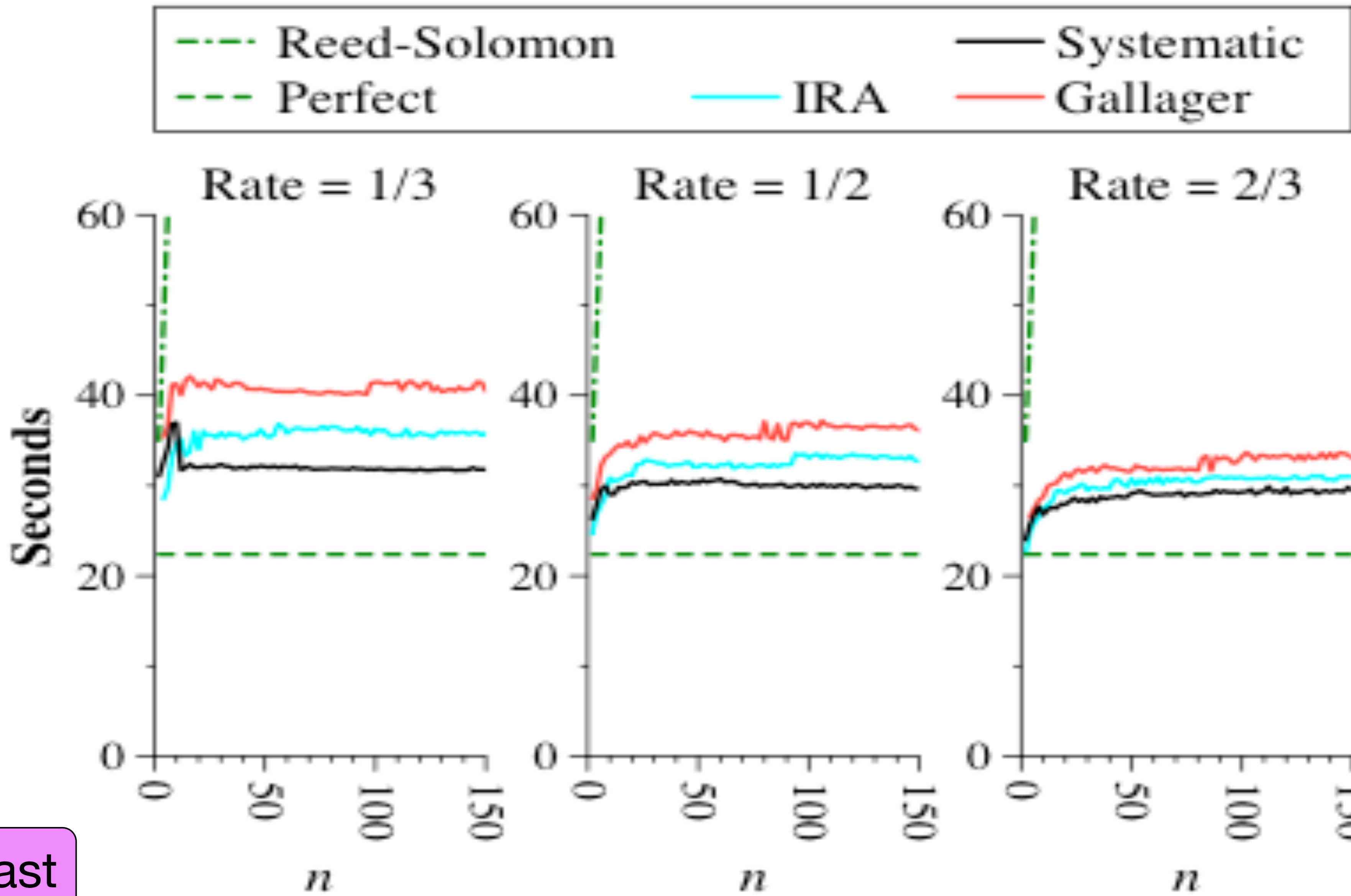


- It Depends:
  - Some bad
  - Some Good
- Open Questions:
  - Do research on convergence!

Q5: Compare with Reed-Solomon?

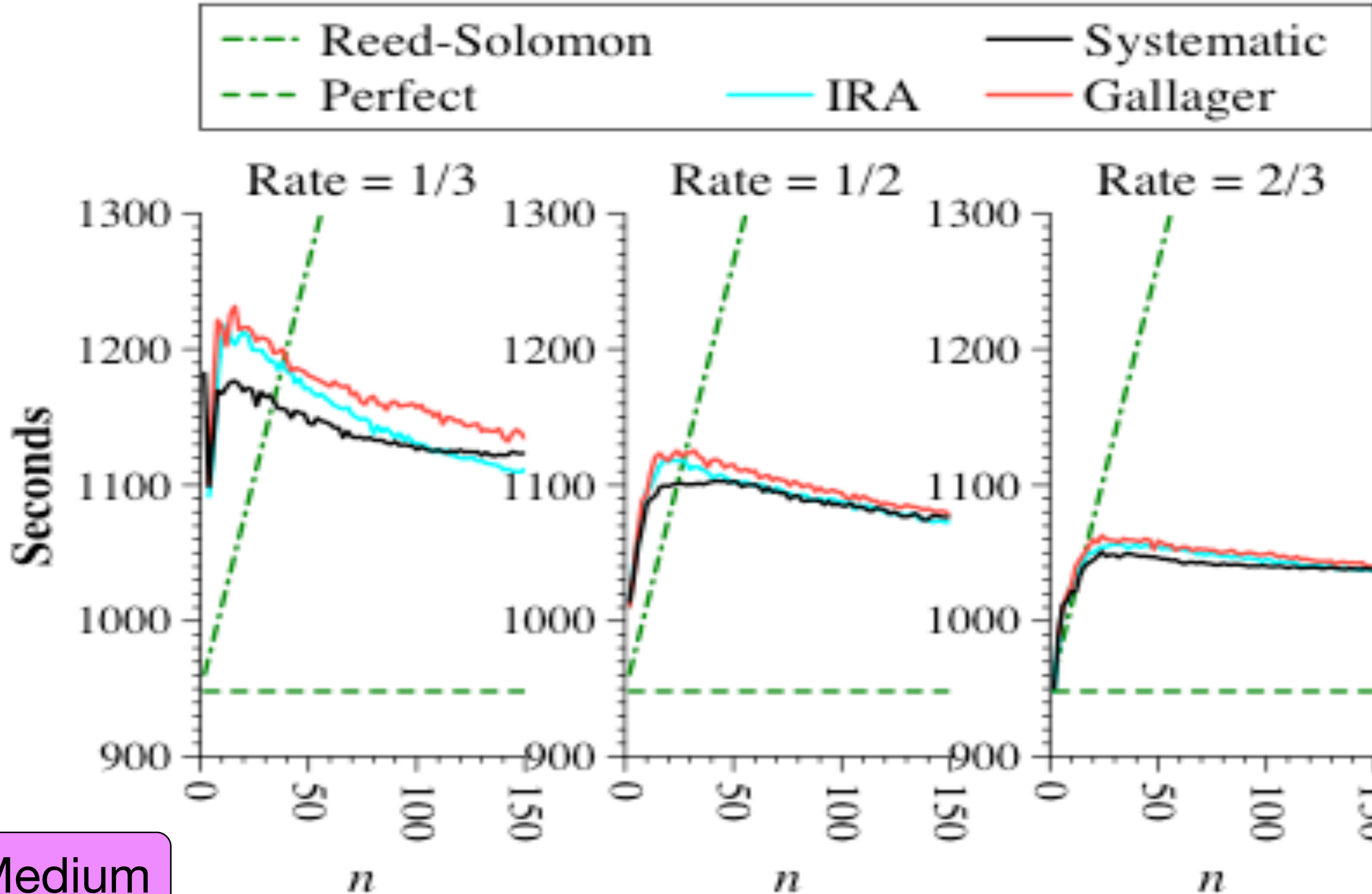
- Using IBP, measured wide-area download speeds to three clients:
  - *Fast*: UT wired: 45.8 MB/s (Megabytes per second)
  - *Medium*: UT wireless: 1.08 MB/s
  - *Slow*: Home wireless: 0.256 MB/s
- Measured computation costs on Linux Workstation
  - S_{xor} = 637 MB/s
  - S_{GF8} = 218 MB/s
  - S_{GF16} = 20.2 MB/s
- Projected performance of LDPC & R-S coding.

Q5: Compare with Reed-Solomon?



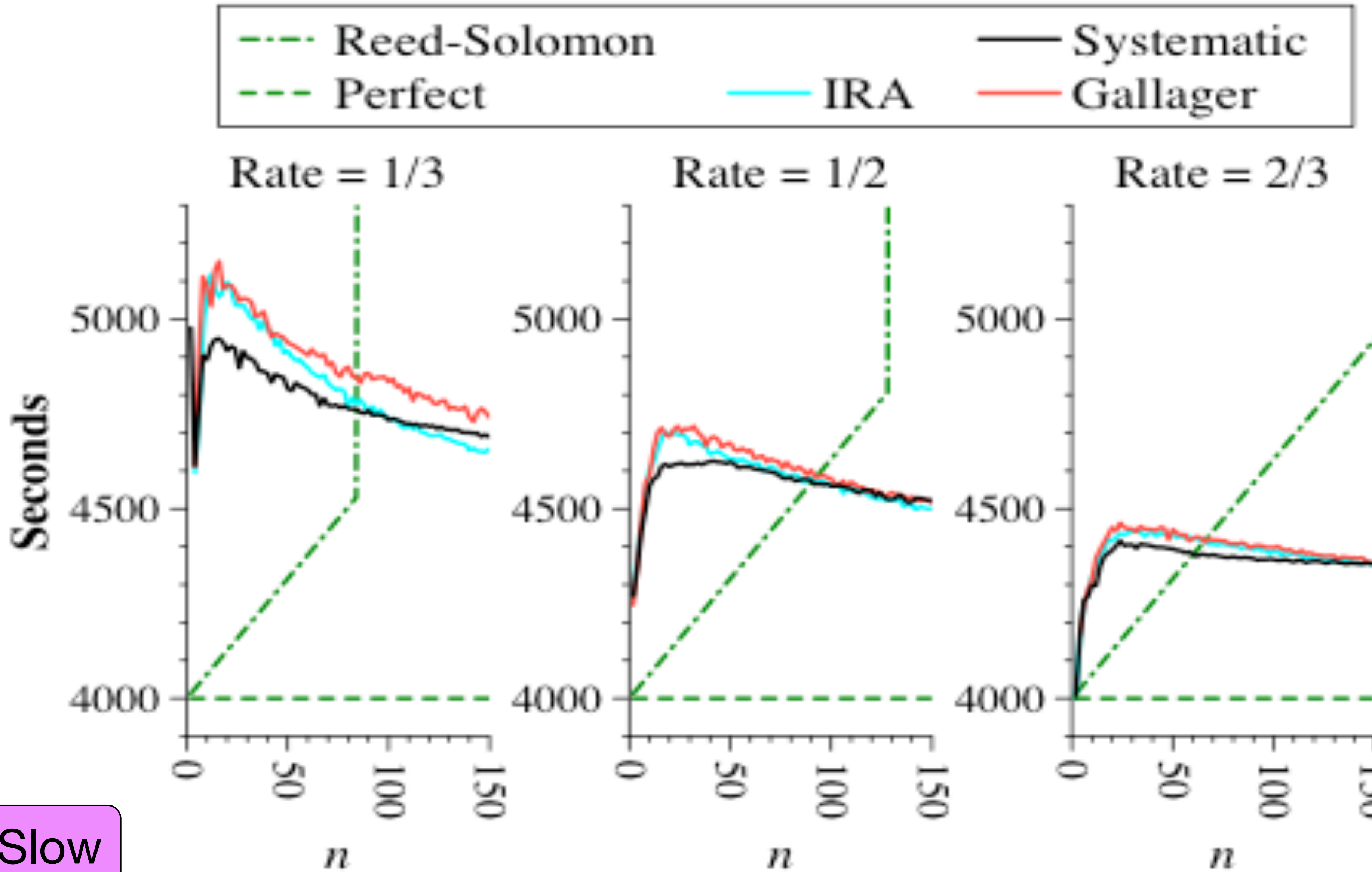
Fast

Q5: Compare with Reed-Solomon?



Medium

Q5: Compare with Reed-Solomon?



Slow

Q5: Compare with Reed-Solomon?

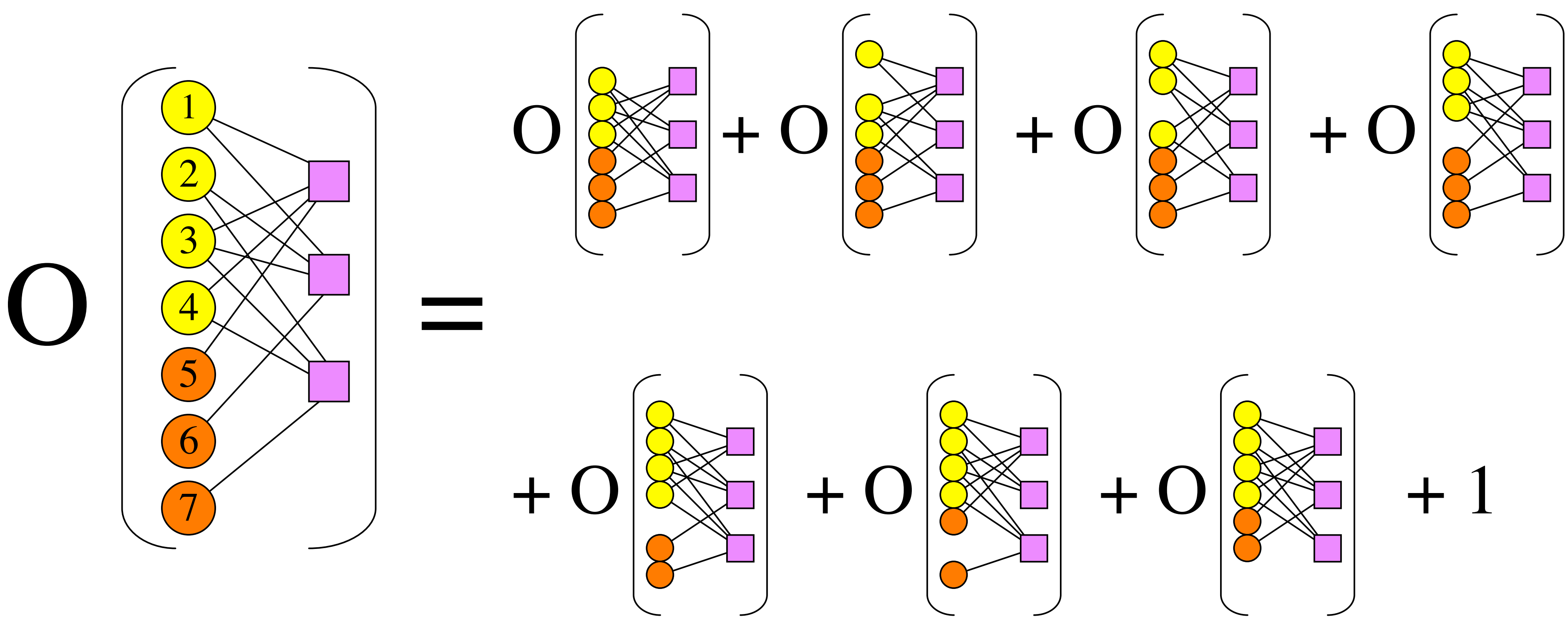
- Sometimes LDPC vastly better:
  - Big *n*, Fast network, Slow computation.
- Sometimes Reed-Solomon vastly better:
  - Small *n*, Slow network, Fast computation.
- Difference in GF8 and GF16 significant.
- Open Questions:
  - Do a better job with all of this.
  - Explore multi-threading, greedy algorithms
    - e.g. [Plank *et al* 2003], [Allen/Wolski 2003], [Collins/Plank 2004].

Conclusions of Study

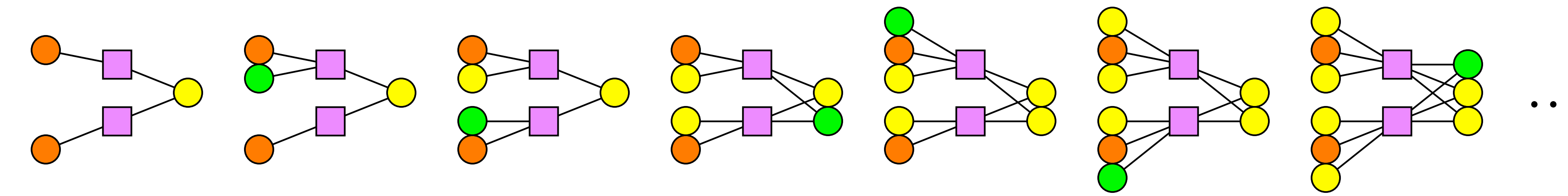
- For small *n*, the best codes arose as a result of the Monte-Carlo simulation. *I.e: □ and □ are very poor metrics /constructors for finite codes*. Theorists need to get to work on better ones.
- Clearly, *even sub-optimal LDPC codes are important alternatives* to Reed-Solomon codes. We need more analysis & parameter studies.
- For serving the needs of wide-area storage system developers, *this area is a mess!* Coding & graph theorists need to get to work on it!

Recent Work: #1. Small, Optimal Codes

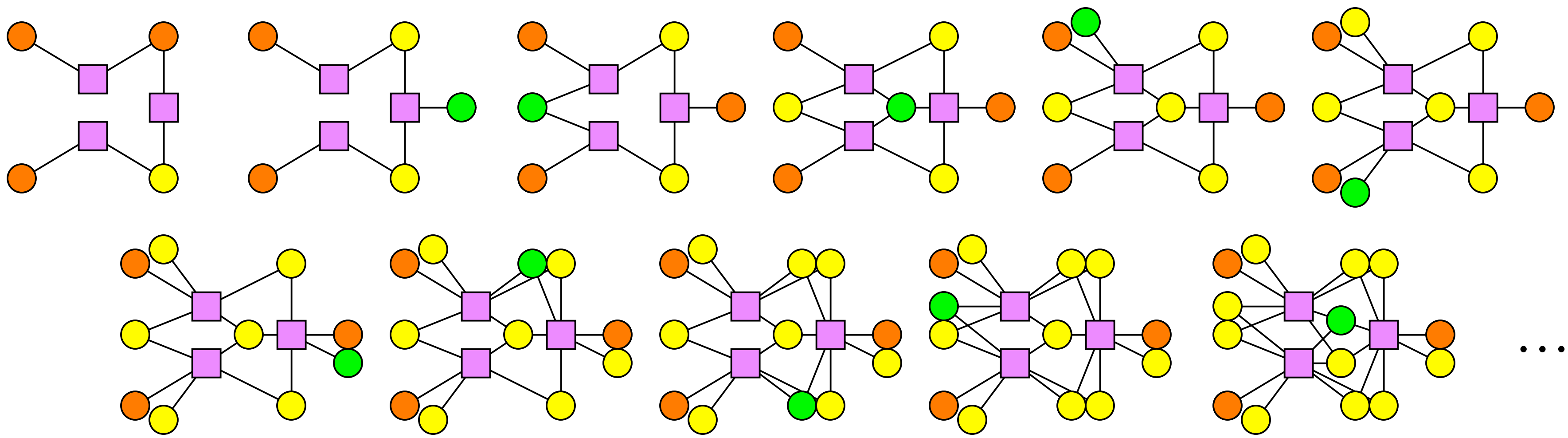
- Use simple enumeration to find the best small codes.
- Calculate overhead recursively:



Recent Work: #1. Small, Optimal Codes



$m = 2$



$m = 3$

Two trends: Balanced node types, incremental graphs.

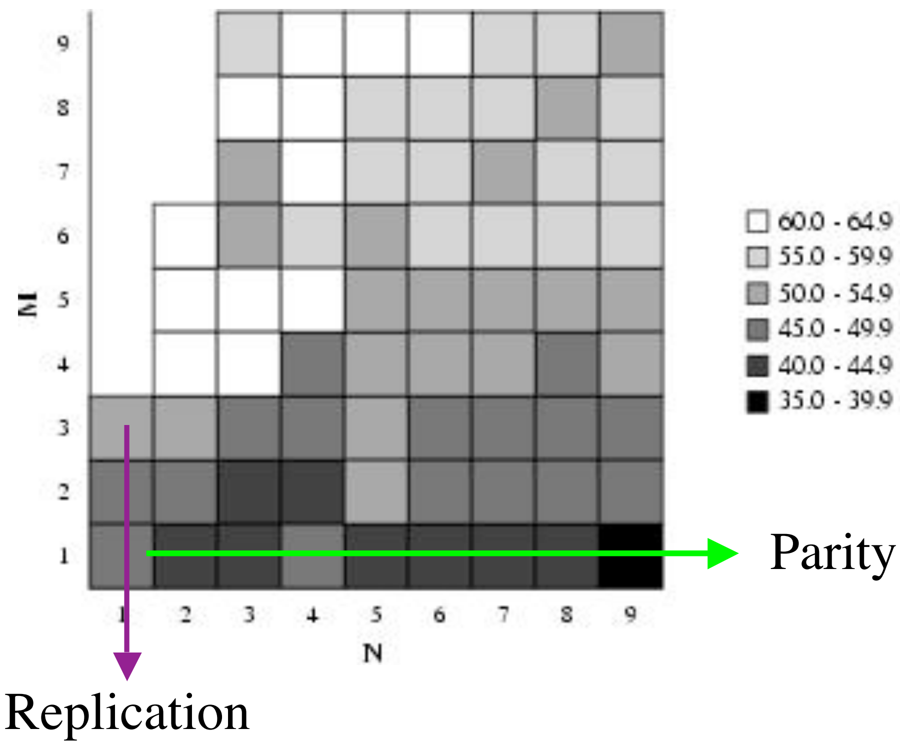
Recent Work: #1. Small, Optimal Codes

Open Questions:

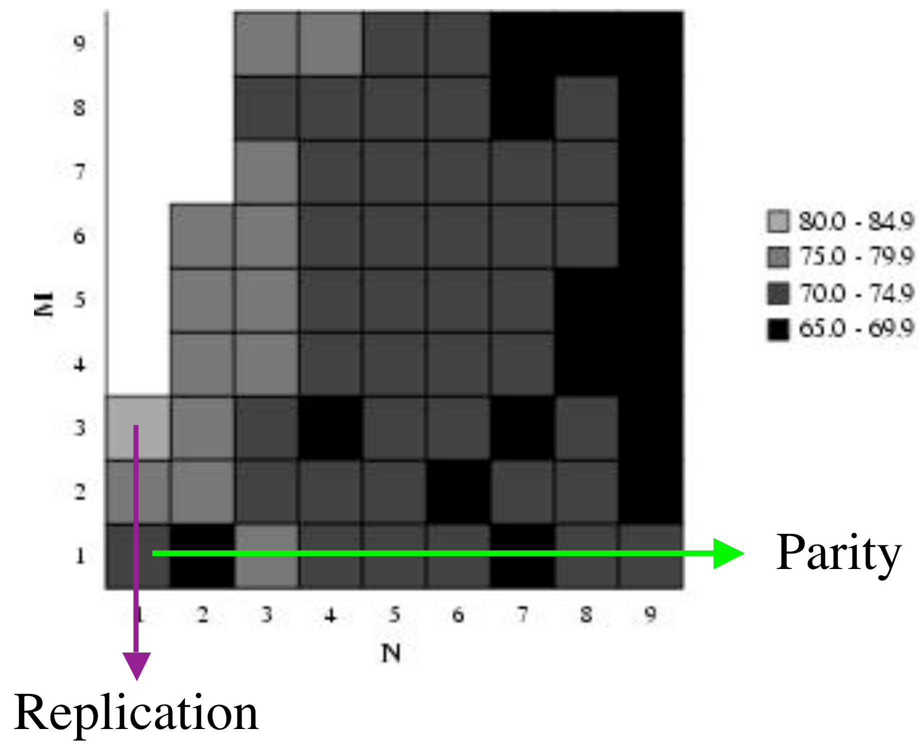
- Does this pattern continue for larger *m*?
- Can we use optimal graphs for small *m* to construct graphs for large *m* (more natural)?
- Can we generate good graphs for large *m* & *n* in an incremental manner?
- Can we prove anything?

Recent Work: #2. R-S Downloads

Bandwidth of downloads with 30 threads and Reed-Solomon coding.



Blocks splattered across 50 servers



Blocks stored in distinct network regions.

All About Erasure Codes:
- Reed-Solomon Coding
- LDPC Coding

James S. Plank

Logistical Computing and
Internetworking Laboratory
Department of Computer Science
University of Tennessee

ICL - August 20, 2004