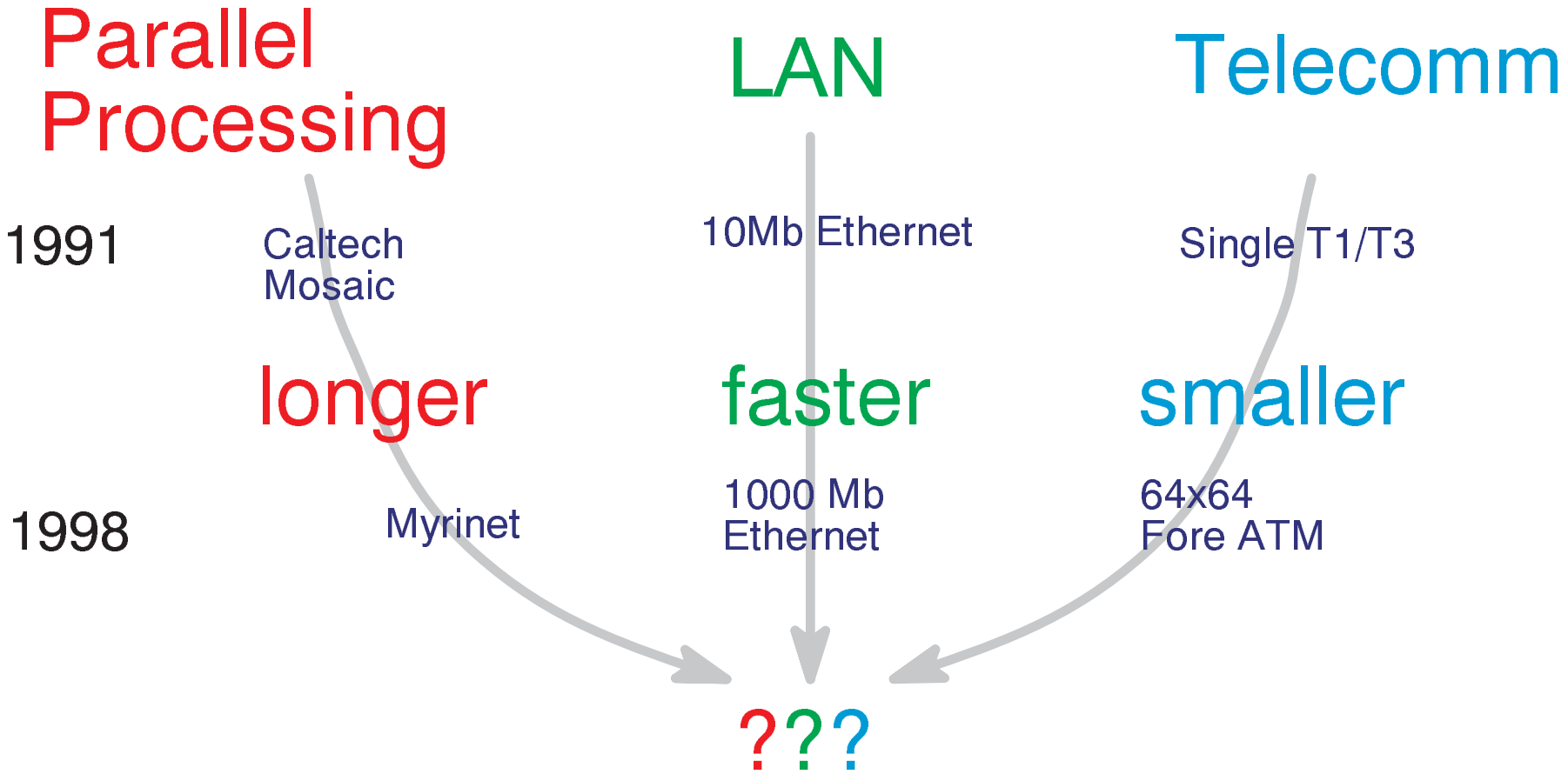
# Network Performance

Richard Martin

9/21/1999

# Context

- 1990s: silicon revolution powers convergence



# Motivation

- Result: Explosion of new networks and apps
  - Build your own parallel computer <$30K
  - Wide area goes beyond telnet, FTP
- Huge Design Space
  - Software and Protocols (AM, BIP, Unet ... )
  - Hardware (Shrimp, Tempest, Myrinet ...)
- Need framework for Systematic Evaluation
  - defining what's better/worse and why?

# Motivation

- How to evaluate a system?
- Micro-Benchmarks
  - Linear models
    - Round Trip Time
    - Bandwidth models
  - LogP model

  - Q: Micro-benchmark distortion?
    - (e.g. MIPS vs SPEC vs TPC) ?
- Application sensitivity to the network
  - Which Applications? What Models?
  - Whole other body of work ...
- Communication Semantics?

# Outline

- Intro
- Parallel Programming Models
  - PRAM
  - Network
  - LogP
- Other communication Models
  - Hockney
  - Bandwidth
- LogP μ-benchmarks
- Pipelining Models
- Communication Semantics

# Parallel Programming Models

- Purposes of PP Models
- Decomposition
  - How to break problem apart into separate data/control
- Orchestration
  - Getting everything to the right place at right time
- Evaluation
  - How well will it run on a real machine?
  - Is this algorithm "good" for under some definition of speedup?

# Parallel Machine Dilemma?

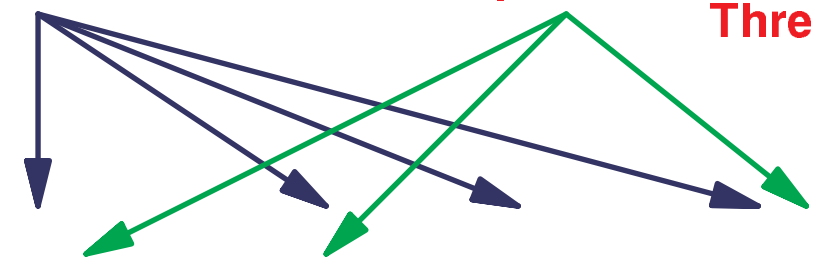
Message Passing

Shared Memory

Array Programming

Abstraction

**MPI** **Sockets** **Split-C** **Posix Threads** **HPF** **Matlab**



Run-Time OS layers

U-Net Kernel TCP VMMC Sys-V Shared Mem RTL?

Communication + Control

Myrinet Shrimp Origin MMX

Physical Machine

# History: Parallel Prog. models

- Communication aspect of the machine
- PRAM Models
  - Processors and Shared Memory
- Network Models
  - Nearest Neighbor communication
- BSP Model
  - "superstep" communication
- Array-Based models

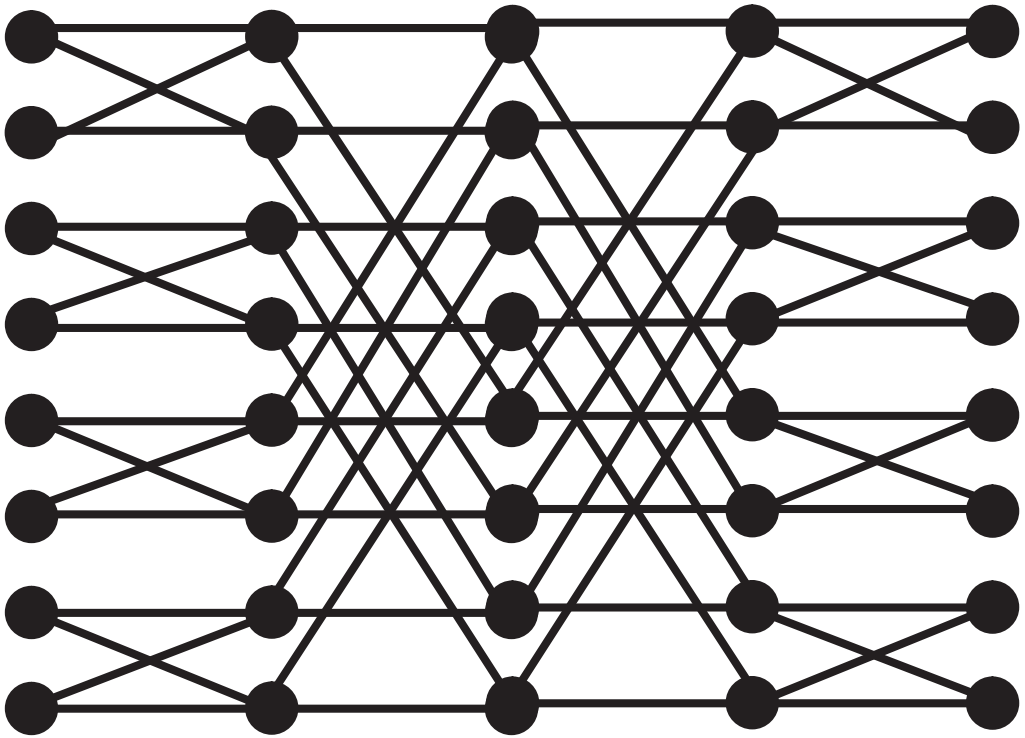
- All models duals to some extent!
  - work for one => straightforward conversions

# PRAM Model

- LogP paper slamed as unrealistic
- Often a good starting point
- Lots of variations
- Too many loopholes!
  - Lots of unrealistic algorithms
- Locality Management still key on real machines

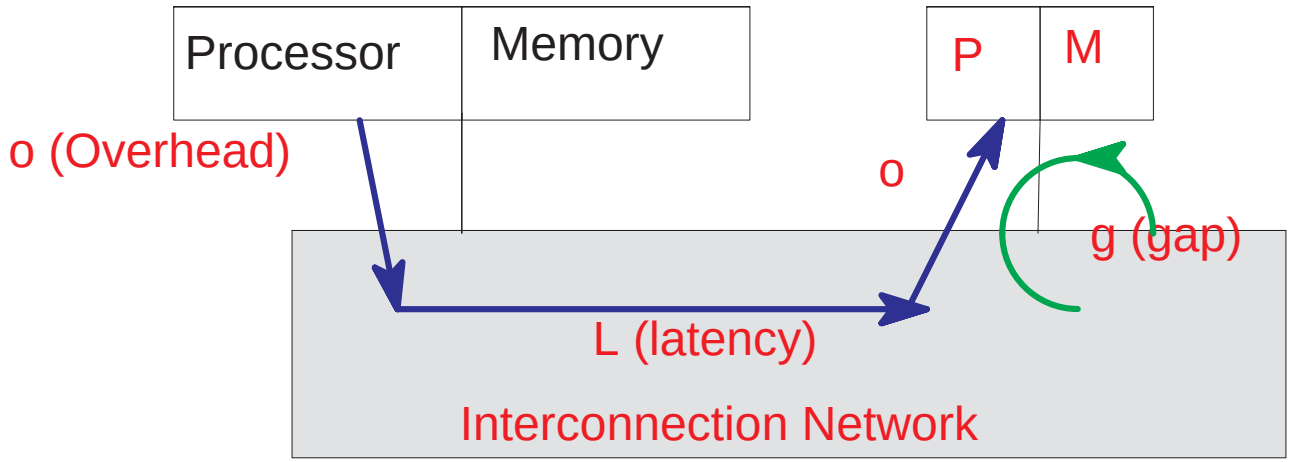
# Network Model

- Eg XYZ (sorting, MM)  on the hypercube
- Map cube/butterfly to some other network
- Mostly Artistic
  - But useful in practice for partitioning!



Sorting Network

# LogGP model




- overhead: time spent by processor
- Latency: time to cross interface+fabric
- gap: min time between sucessive messages
- bulk Gap: time per byte for long transfers
- Processors

# LogGP model

- Way to reason about machine performance
  - Good for evaluation....
    - If I partition problem like X, it will run in time Y
  - Good as an architect's model
    - Machine must support multiple programming models (like machine ISA)

- Partitioning, orchestration?
  - PRAM very good
  - Array models good
  - Not too hard to map to a LogP analysis.

# LogGP Model

- General Network Model
  - Not just parallel machine performance
- E.g. Web-server
  - Overhead vs. latency vs. gap vs Gap
  - Which one should be optimized?
- Optimize:
    - Time of 1 request (response time)?
    - Throughput of many small requests ?
    - Throughput of large requests?
    - Combination (Response time vs. Throughput)?

# LogGP Model

- Captures many classes of performance
- Time of 1 request
  - o + L+o+o+L+o
- Throughput of many small requests
  - o and g
- Big requests
  - G
- Combination
  - g as a f(requests/second)
- E.g. gathercast in LogGP terminology
  - Transformation of g into G with cost L'

# Other Communication Models

- Hockney linear time model
- Start-up cost ($T_0$) + per-byte cost ($R_\infty$)
- Time to send a message of size B is:

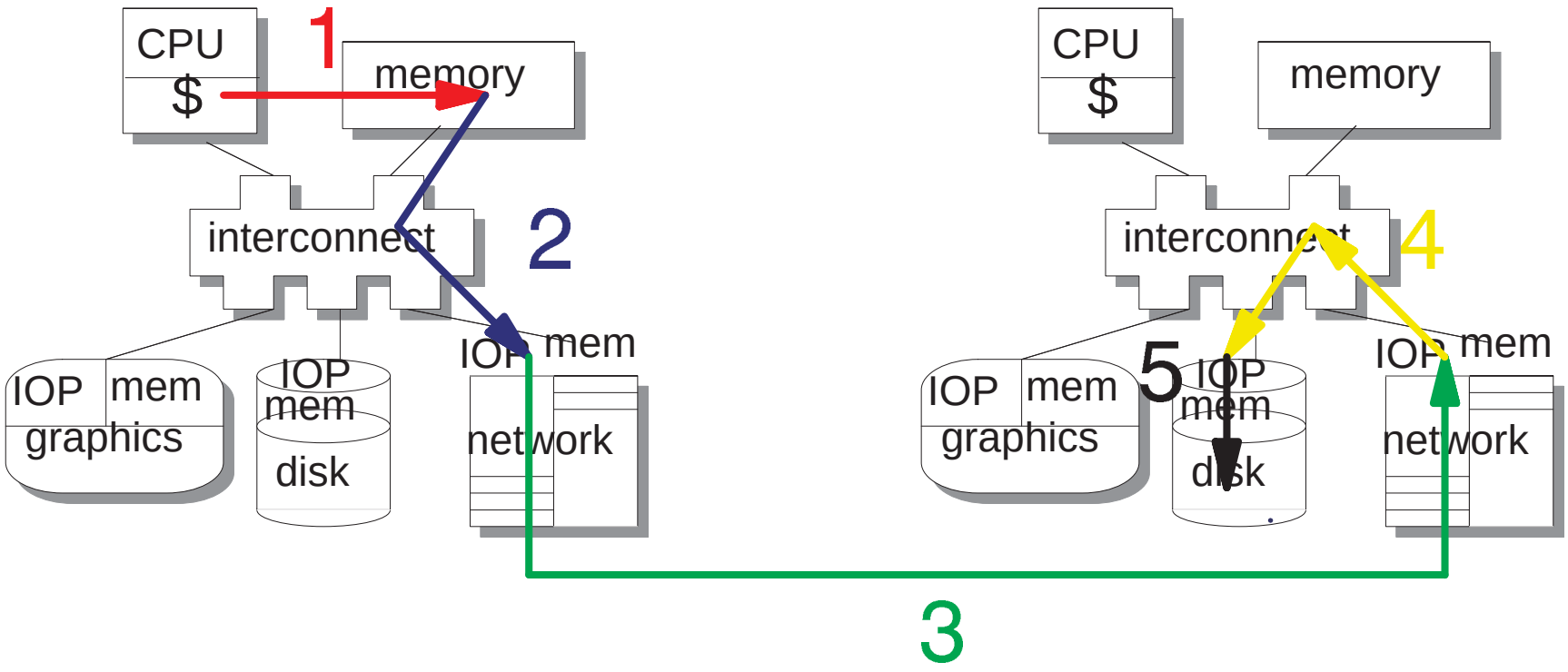
$$T_B = T_0 + \frac{B}{R_\infty}$$

- $T_0$ like LogP overhead
- $R_\infty$like LogGP Gap
- Linear fit in practice?

# Bandwidth Model

- Seen a lot in networking lit.
- Show bandwidth as F(packet size)
- somewhat useful.
- Often abused

# Why Study these Models?




- View path as a communications pipeline
  - Even to disk!

# Abstract Models

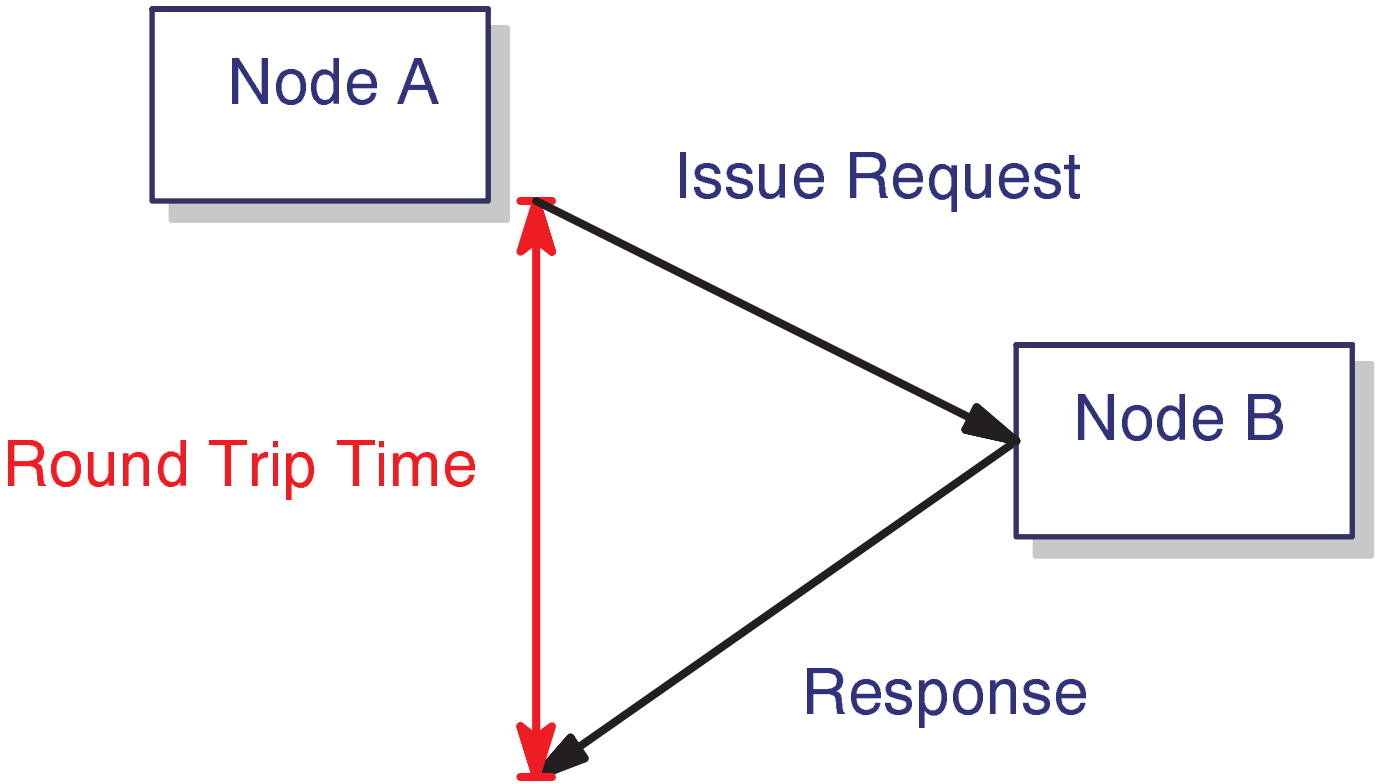
- How to reason about performance
- LogGP processor-centric
  - o, everything else in g,G, L
- Pipeline model captures more of the system
  - Each stage modeled by an "overhead" and "gap" term
- When to use each depends on what you care about.
  - Application vs. system

# Microbenchmarking

- How to get L,o,G,g?
  - 1. direct measurement
  - 2. black-box
- Issue sequence of probes
- Derive parameters from response
  - logP signature
  - First probe is a Round-Trip Time = 4o+2L
  - Second probe sends burst of N messages, and remote side does N replies. For large N, T/N = g
- How to extract o and L from the RTT?
  - Solution: add extra overhead, D to reach msg

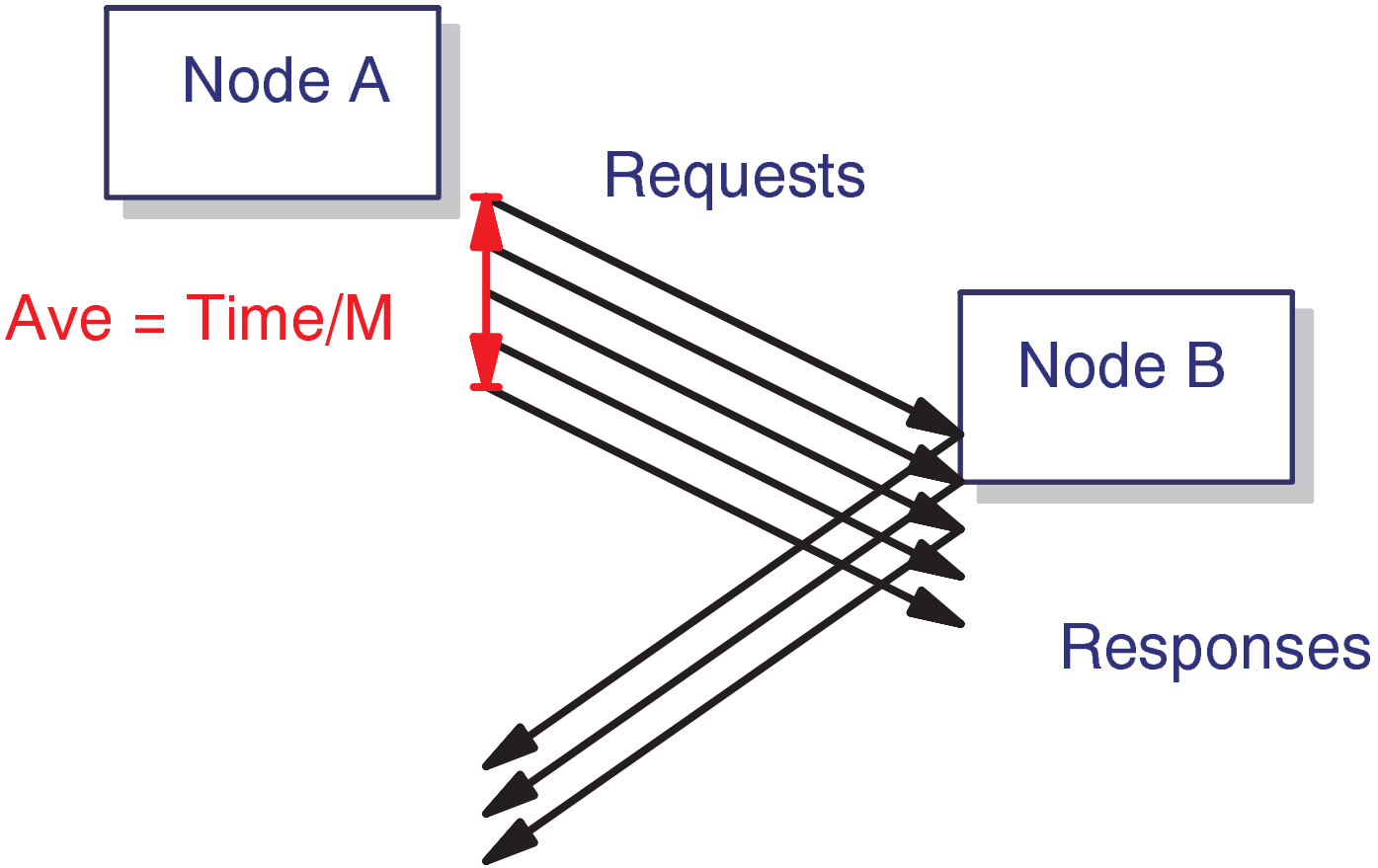
# First Probe

- Compute Round Trip Time

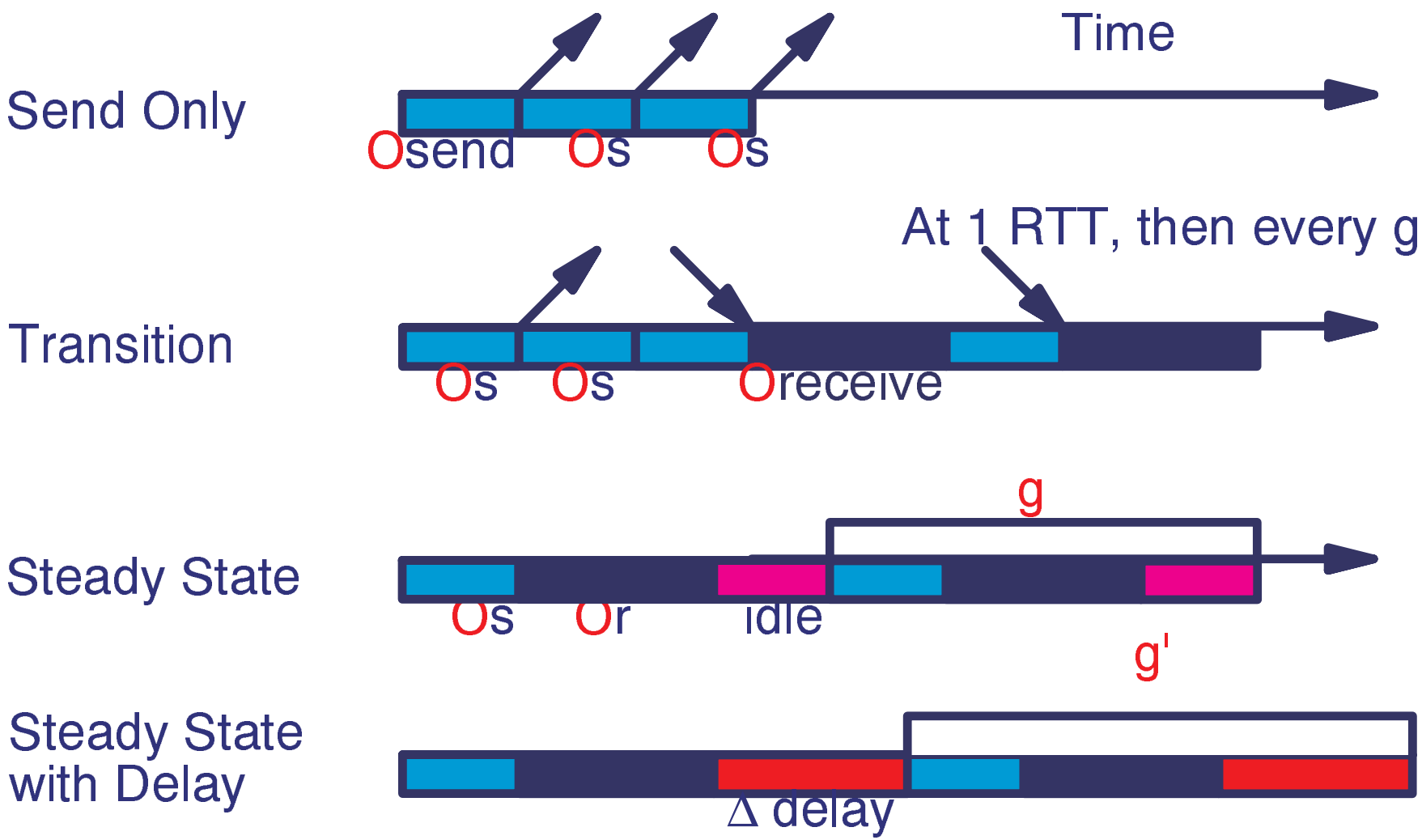


# Second Probe

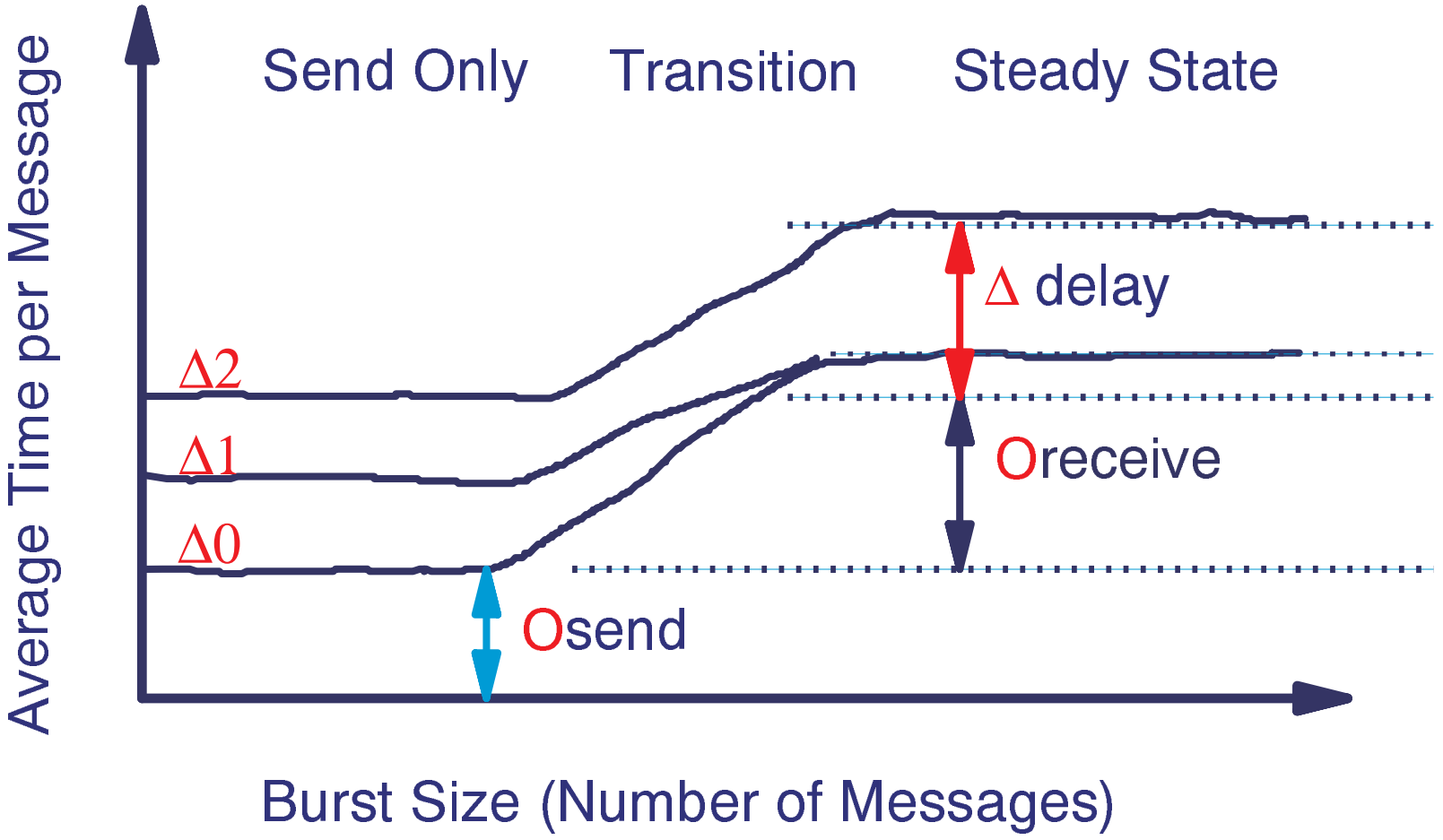
- Issue Sequence of M messages
- Compute Average Message Cost



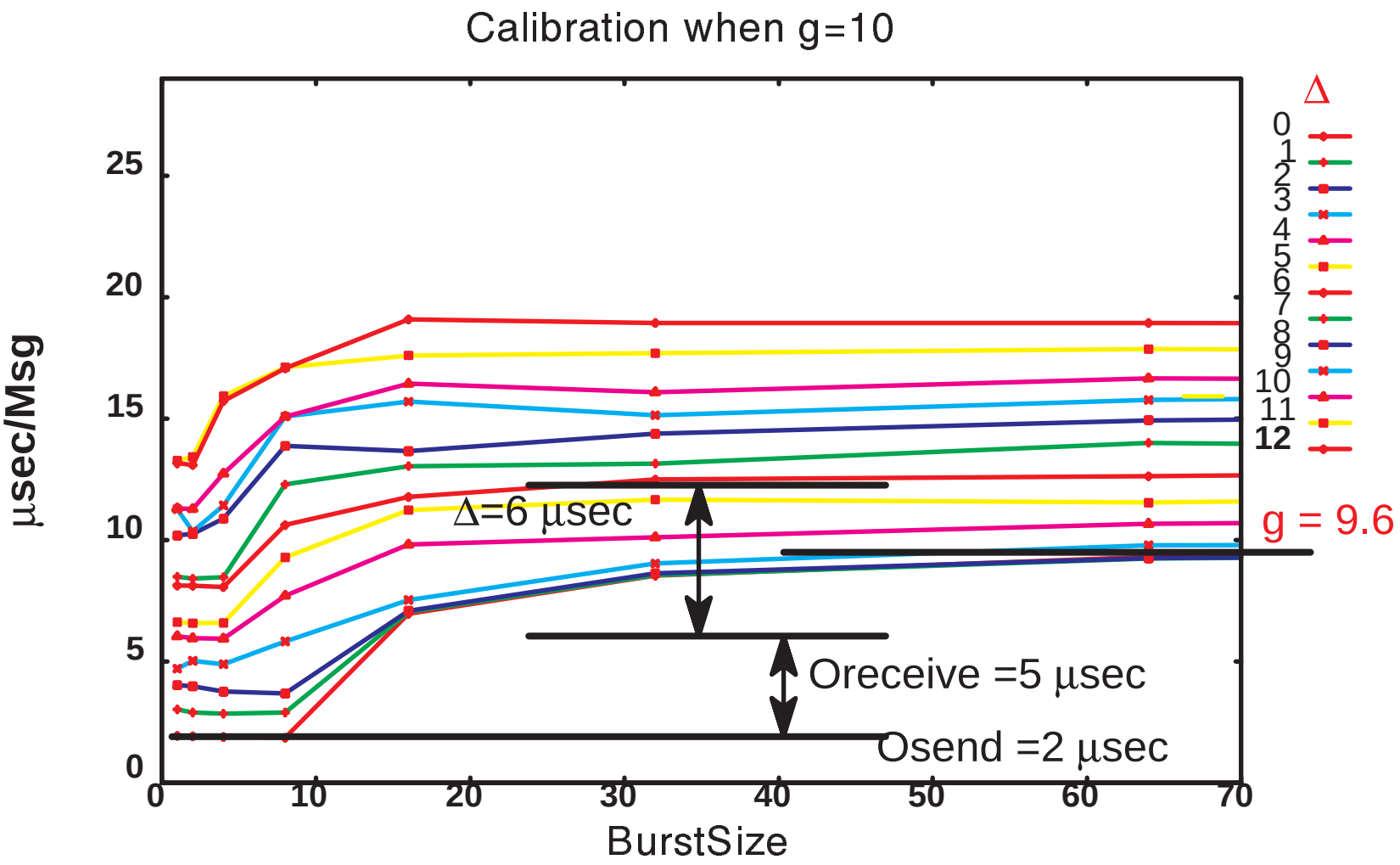
# Regimes of Operation for M



# Abstract Signature



# Actual Signature



# Pipelining

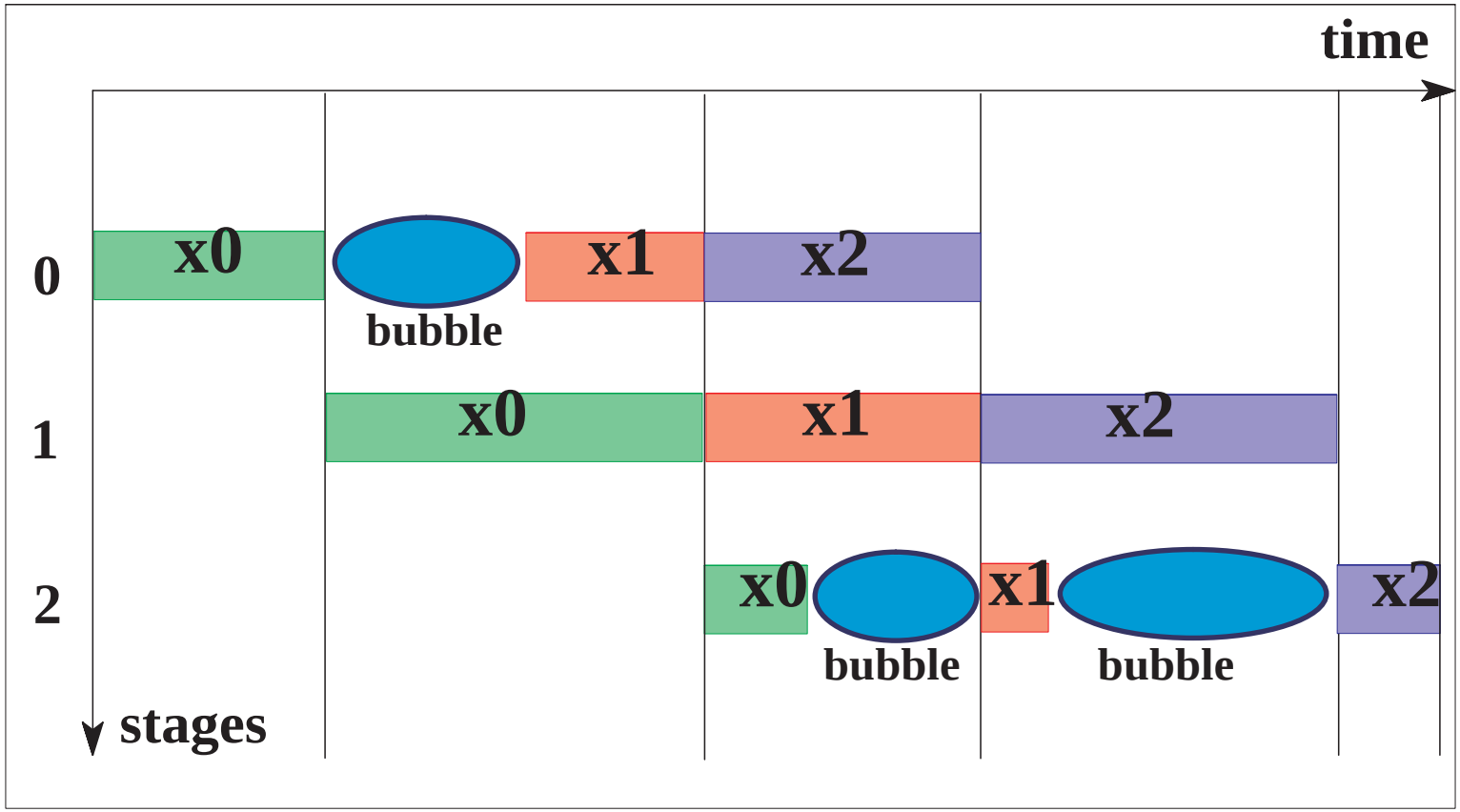
- What's the minimum time to move a 16 Kb page to a memory server?
- Observations:
  - At time scales of fast networks, sending all 16Kb at once incurres too much store and forward delay
  - Should send part of page first, then remainder
- Key insight:
  - Want to keep entire network pipeline "busy" for the transfer.
- Q: How to optimally fragment page?

# Pipelining Model

- Want to send a packet
  - can break it down into fragments
- Network has a number of pipeline stages
- Each stage is defined by:
  - per-fragment startup cost
  - per-byte cost
- Each stage is store-and-forward

- Question:
  - how to break down the packet to minimize latency of the entire packet?

# Pipelining Model



# Notation

- Packet size is B bytes long

- K fragments
- Fragments $X_0+X_1 \ldots X_{(K-1)} = B$

- Stage i defined by fixed overhead cost + per-byte cost
  - $(g_i, G_i)$
- Bottleneck stage is
  - $(g_b, G_b)$

# Algorithms

- Linear (Integer) Programming
  - Closed form solution, gives true lower bound
  - Use in Practice?
    - Look-up tables (e.g. BIP)
- Fixed-Sized Fragmentation
  - Easy to do in practice
  - Keeps re-construction overheads low
- Variable Sized Fragementation
  - Ramp-up/down theorems
  - Practice?
    - Lookup table

# Linear Programming

- $T_{(i,j)}$ : Time of $i^{th}$ fragment to cross $j^{th}$ stage

- Si: Size of fragment i
- K = number of fragments
- L = total number of stages
- Free Variables are $T_{(i,j)}$ and Si

- Create a system of linear inequalities and minimize time of last fragment $T_{(k-1,j-1)}$

# Linear Programming

- Time of fragment i must be greater than previous packet in both stage and pipeline dimensions:

$$T_{(i,j)} \geq T_{(i,j-1)} + g_i + S_i G_i$$

$$T_{(i,j)} \geq T_{(i-1,j)} + g_i + S_i G_i$$

# Linear Programming

- Bound the time of the first packet

$$T_{(1,j)} \geq \sum_{r=1}^{k} (g_r + S_r G_r)$$

$$T_{(1,0)} \geq 0$$

# Fixed Sized Fragmentation

- Total Time = Time before bottleneck + Time in bottleneck + Time post bottleneck
- = Time in bottleneck + everything else

$$T = g_b + G_b + \sum_{i \neq b} (g_i + G_i)$$

- Find min: take derivative with respect to K and set to zero
- Result:

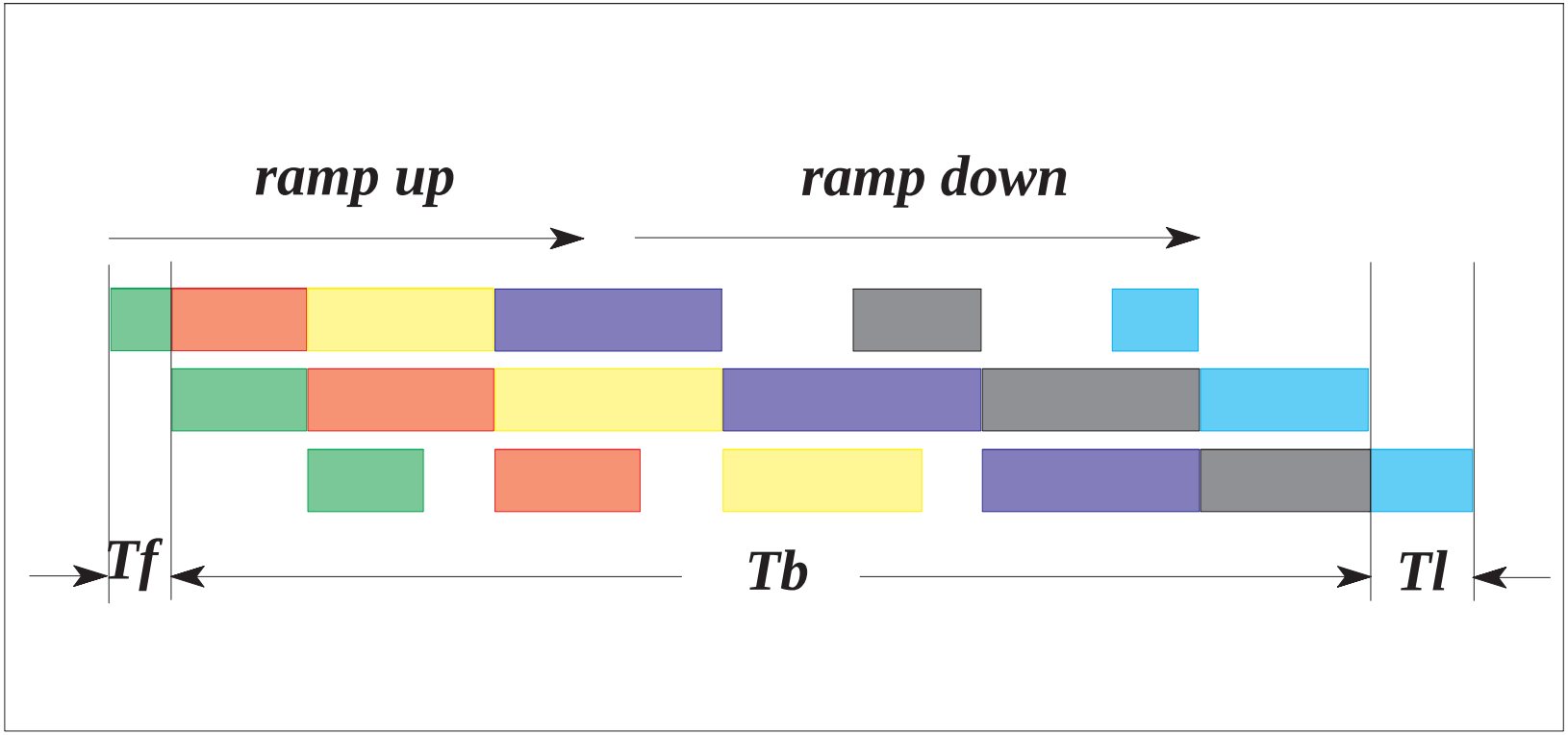
$$K = \sqrt{\frac{B * \sum_{i \neq b} Gi}{g_b}}$$

# Fixed Sized Fragmentation

- Implications:

- No best fragment size for all packets sizes B
- Balance of bottleneck overhead + Bandwidth of other stages
- Overheads in non-bottleneck + bandwidth of bottleneck to not affect performance

# Intuition: Ramp Up/Down




- Gradually increase fragment sizes to keep bottleneck busy
- Reverse also true (ramp down)

# Semantics

- Performance at any cost?
- What about semantics and properties of the message layer?
  - Deadlock/livelock avoidance?
  - Unbounded memory usage?
  - Multiprogramming?
  - Fault Models?
  - Concurrent access?
  - Join/leave semantics?
- Project idea: An investigation of of these issues with respect to performance