Performance Analysis Model

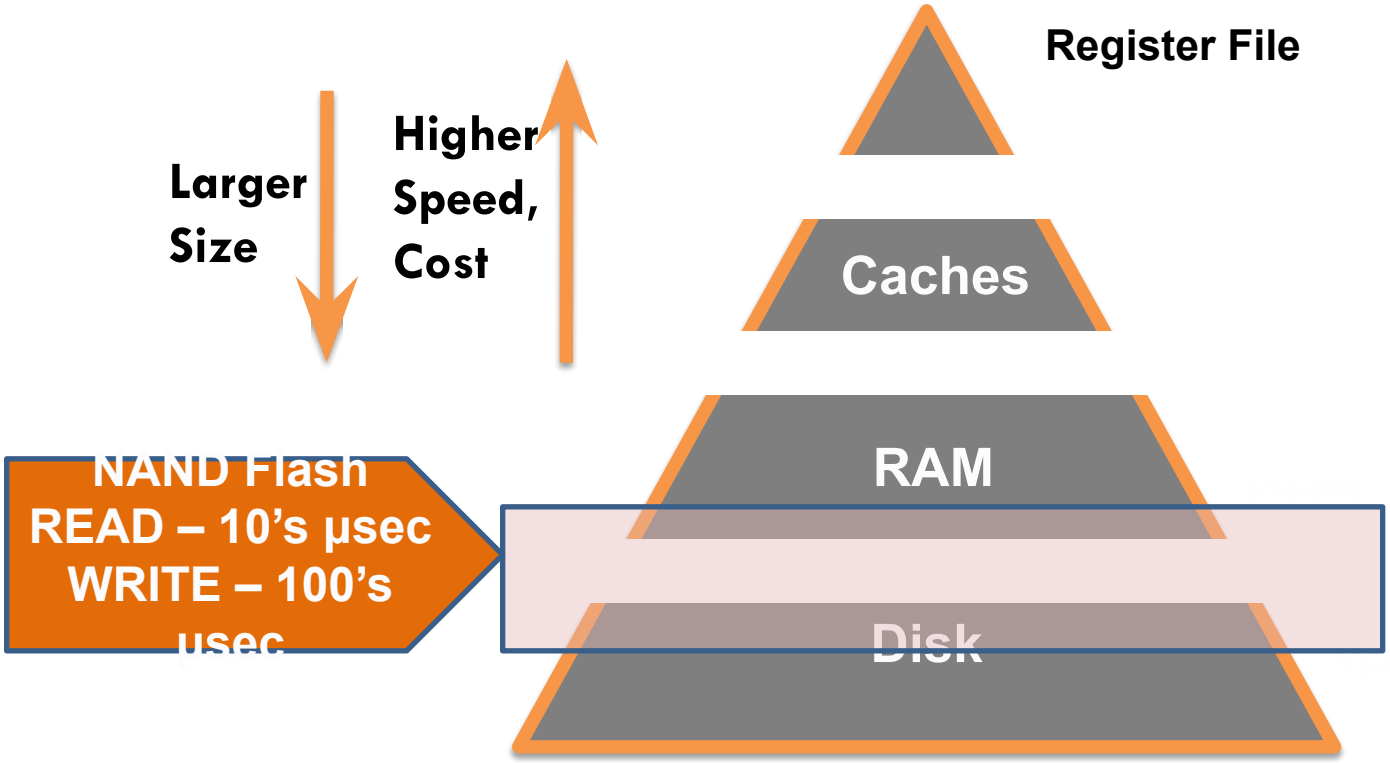
Jihong Kim
Dept. of CSE, SNU

Performance Metrics

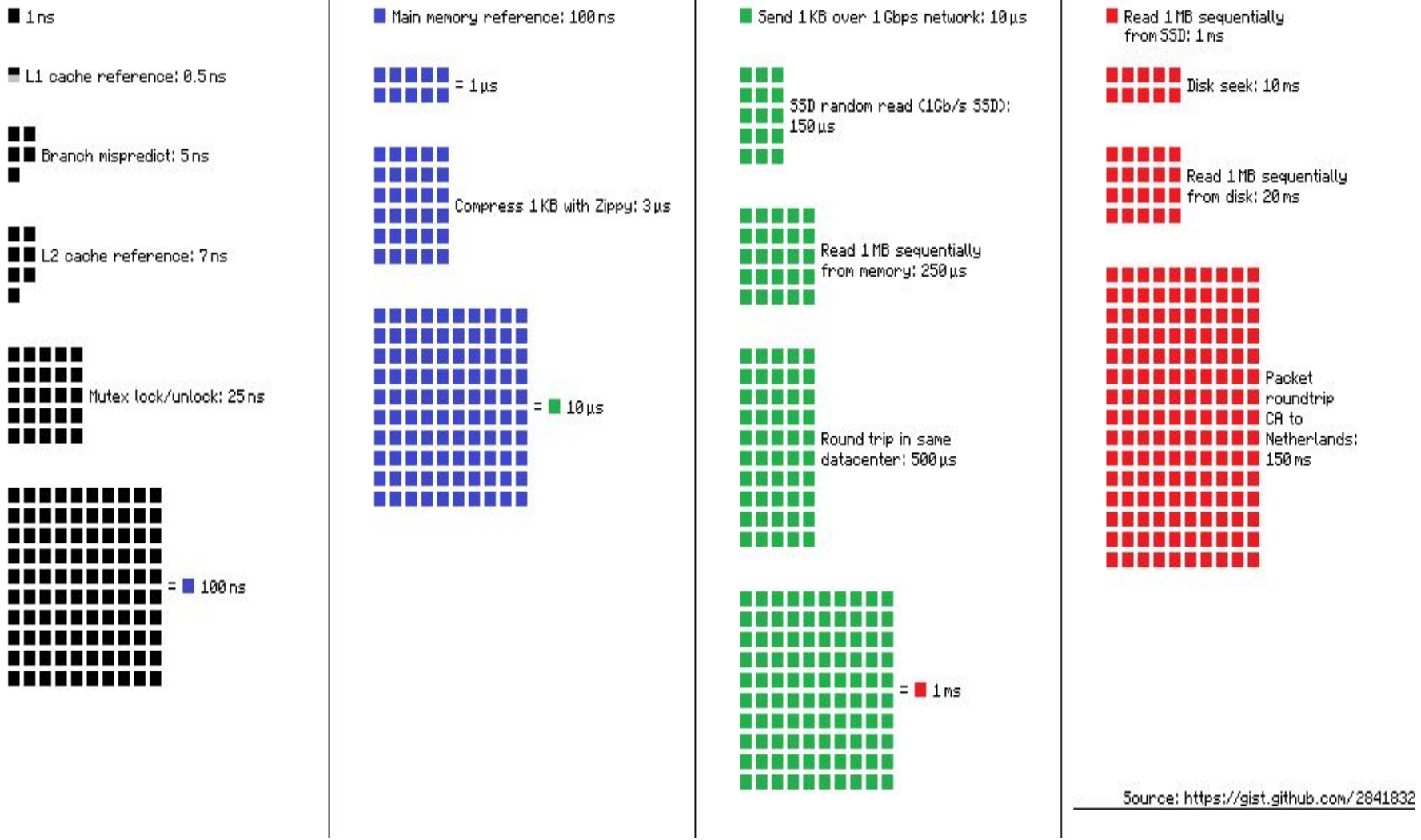
- Throughput
- Response Time
- Capacity
- Reliability
- Cost
- …

- Used for CPU, Memory, SSD, Network, ….

Memory Hierarchy



Latency Numbers Every Programmer Should Know

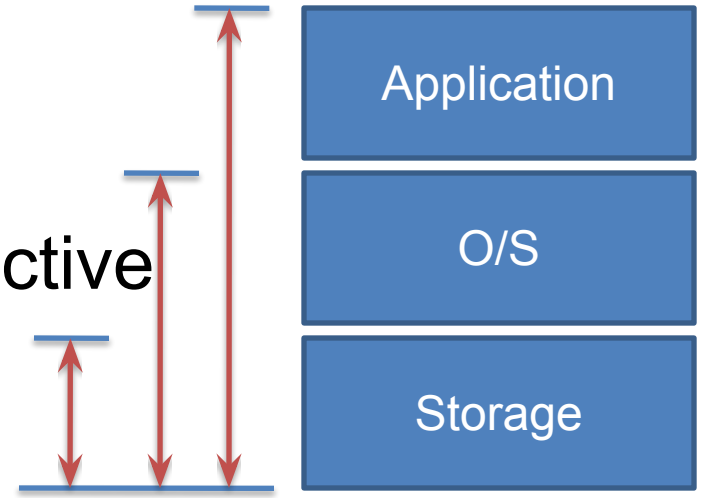


Throughput

- IO rate
  - IOPS (accesses/second)
  - Used for applications where the size of each request is small
  - e.g. transaction processing
- Data rate
  - MB/Sec or bytes/Sec
  - Used for applications where the size of each request is large
  - e.g. scientific applications

Response Time

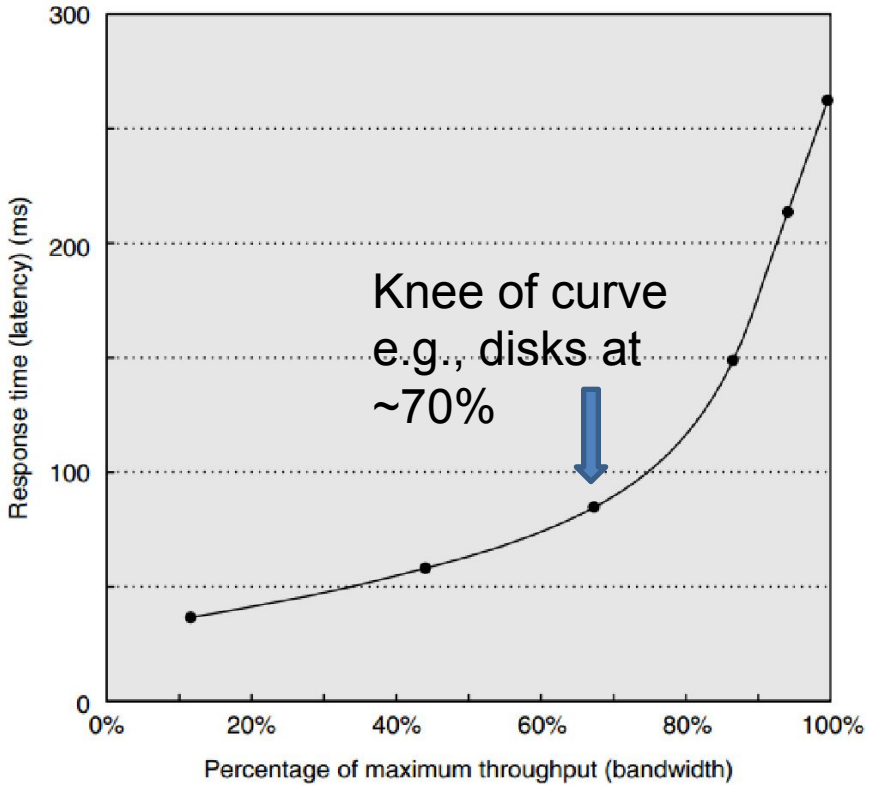
- How long a storage system takes to access data
- Depending on what you view as the storage system
  - User's perspective
  - OS's perspective
  - Disk controller's perspective

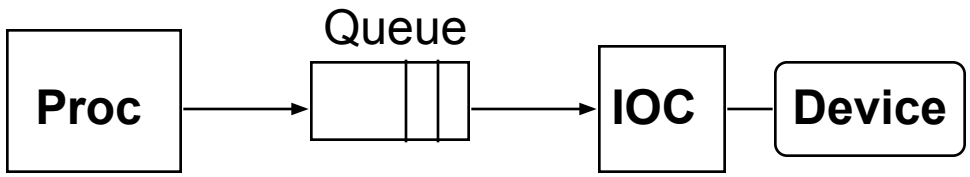


Disk I/O Performance

Metrics:
- **Response Time**
- **Throughput**

Q: how to balance response time & throughput? (e.g., adding more servers)





Response time = Queue + Device Service time

[Hennessy&Patterson]

Performance Analytic Model

- Better to understand key performance factors
- High-level insight on system performance

- Complementary to other performance evaluation methods:
  - Measurement-based
  - Simulation-based

Example Analytic Model

- Original Amdahl's Law
  - Useful in understanding an upper bound on overall speedup (insight with less accuracy!!)

S: speedup

N: no. of processors

f: fraction of sequential part

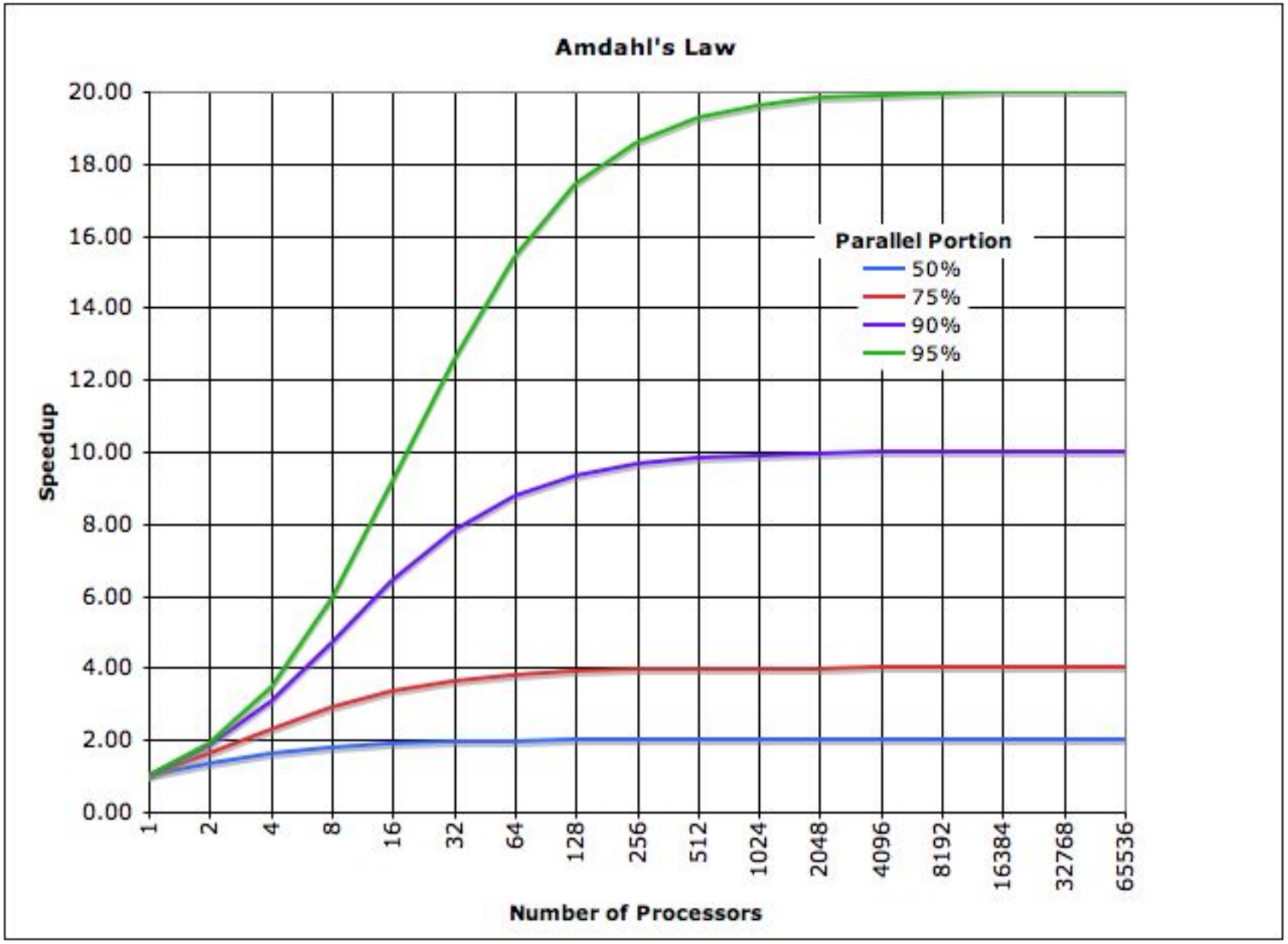
t_p and t_s:
exec. time for parallel and sequential part

$$S = \frac{t_p + t_s}{\frac{t_p}{N} + t_s},$$

Example: 100 processors

$$f = \frac{\frac{1}{S} - \frac{1}{N}}{1 - \frac{1}{N}}$$

- S = 100
  - f = ?
- S = 99
  - f = 1/(99*99) = 0.000102


Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Amdahl's law (in general)

Execution *Time_after improvement*

$$= \frac{\textbf{Execution Time\_}affe}{\textbf{Amount of i}}$$

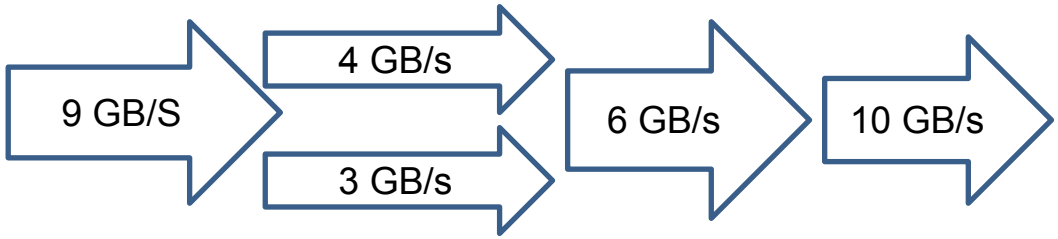
+ Execution Time_*unaffected by improvement*

Bottleneck Analysis

- An upper bound on the throughput of a system from the throughput of its subsystems.

1. The max. throughput of K subsystems in parallel
   = the sum of the subsystem throughputs.
2. The max. throughput of K subsystems in series
   = the minimum of the subsystem throughputs.

Example: Bottleneck Analysis



- Max Bandwidth = MIN(9, (4+3), 6, 10) = 6

- The most expensive subsystem should be bottlenecked.
- Non-bottlenecked throughputs should be decreased to save cost.

Roofline Model

- A visual way to identify performance bottlenecks and to find potential optimization directions
- Bottleneck analysis between

application characteristics and architectural capabilities

$$FLOPS = \frac{\#FLOP}{sec} =$$

$$= Arithmetic\ Intensity \times Bandwidth$$

$$= AI \times BW$$

Attainable FLOPS = min(AI x BW, Peak FLOPS)

application characteristics

architectural capabilities

Arithmetic Intensity = $\frac{\#FLOP}{byte}$

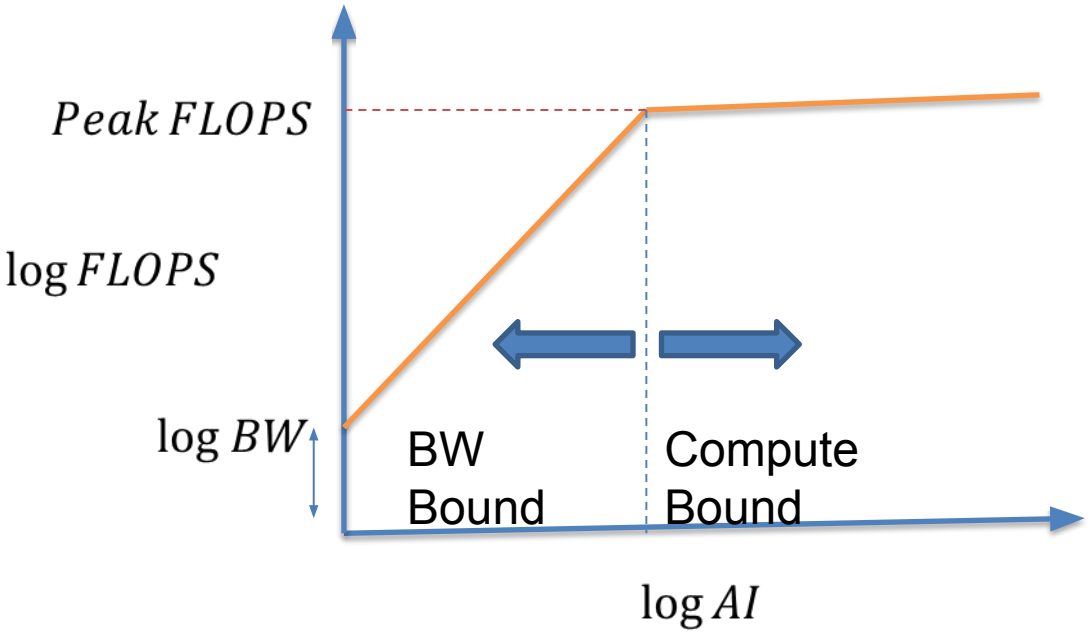
- The number of operations per unit data
- Algorithm specific
- Example AI:
  - Matrix Addition of two N x N matrices
    - Total number of data accesses = 2NN (reads) + NN (writes)= 3NN
    - Total number of operations = NN (adds)
    - $AI = \frac{NN}{3NN} = \frac{1}{3}$ (constant)
  - Matrix Multiplication of two N x N matrices
    - $AI = \frac{(2N-1)\times NN}{3NN} = \frac{2N-1}{3}$ (as N gets higher, AI increases)

Roofline Graph

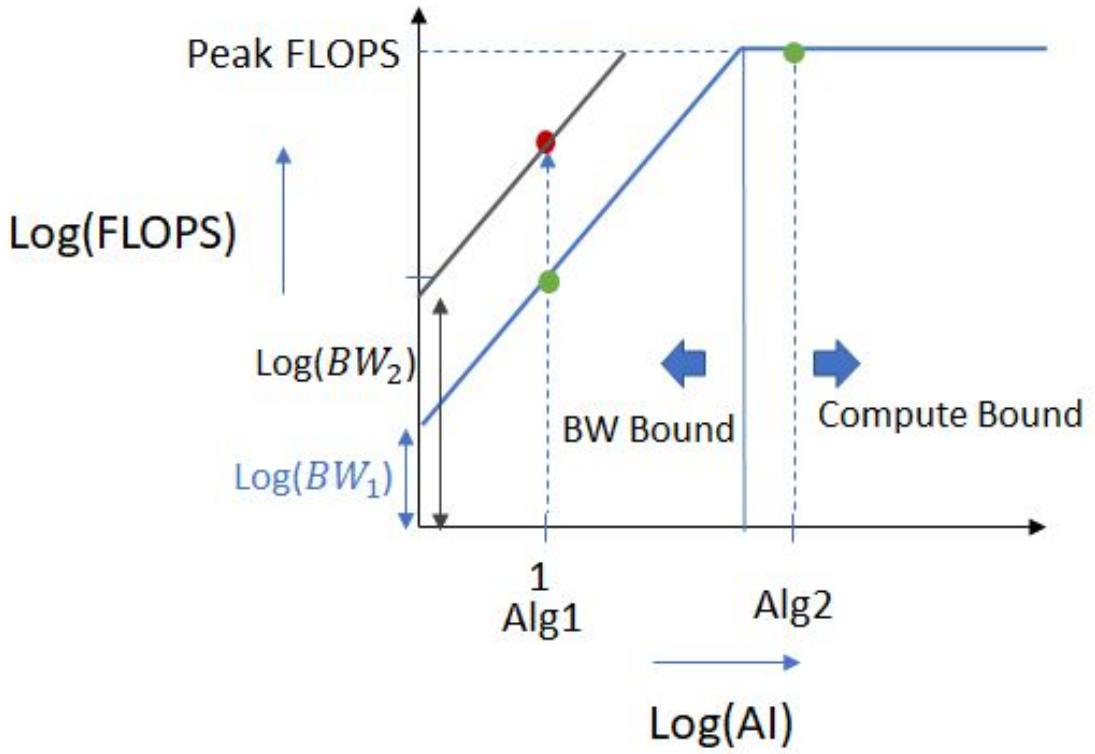
- $FLOPS = AI \times BW$

$$\log FLOPS = \log AI + \log BW$$

$$Y = X + C$$



Optimization Guidelines

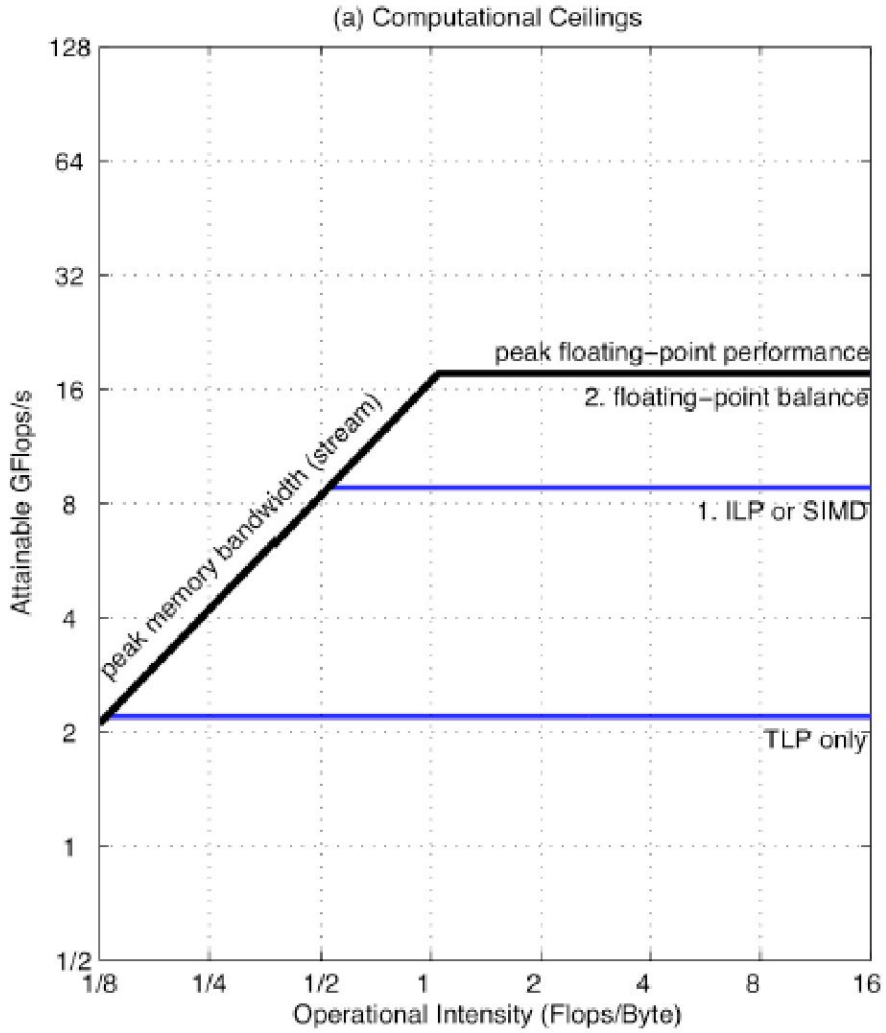


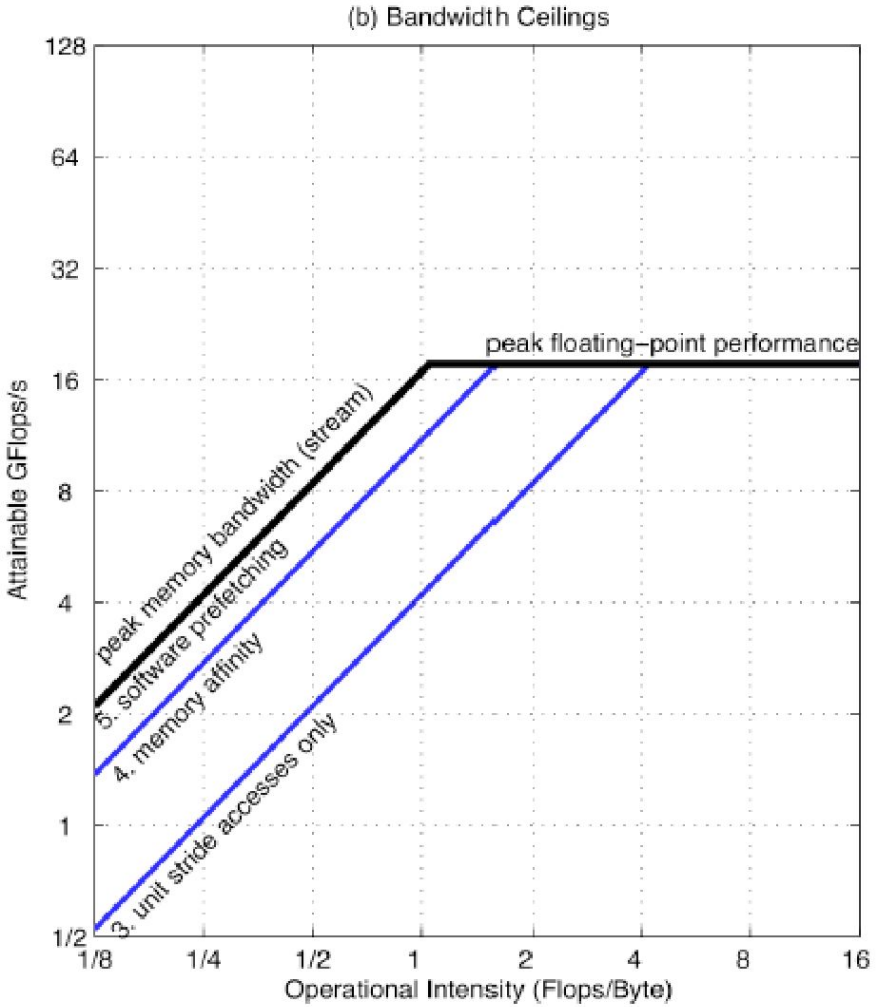
Alg 1: Move Right (higher data reuse)

Move Up (higher BW memory)

Alg 2: Move Up (more powerful CPU)

Roofline Model w/ Ceilings



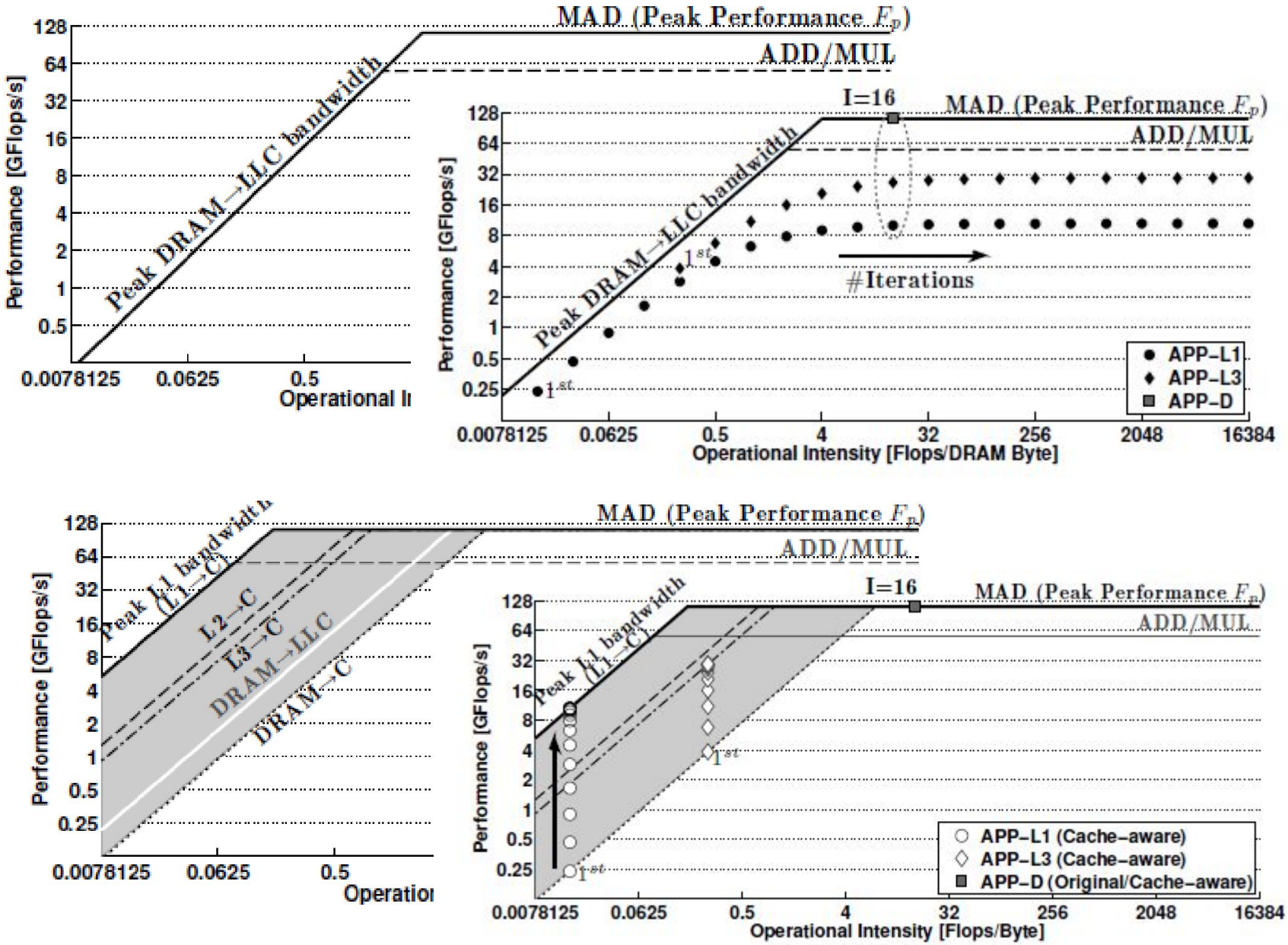


Cache-Aware Roofline Model

- Original Roofline Model
  - AI is computed over bytes moved out of DRAM.
- Cache-aware Roofline Model
  - AI is computed over the total number of bytes moved to the CPU. (That is, L1-cache AI)

AI varies for a given algorithm:
(e.g., different loop iterations or different optimizations)

AI is constant.


MAD (Peak Performance F_p)
ADD/MUL
Peak DRAM→LLC bandwidth
Performance [GFlops/s]
128
64
32
16
8
4
2
1
0.5
0.25
0.0078125
0.0625
0.5
4
32
256
2048
16384
Operational Intensity [Flops/DRAM Byte]
I=16
#Iterations
1st
APP-L1
APP-L3
APP-D
Peak L1 bandwidth (L1→C)
L2→C
L3→C
DRAM→LLC
DRAM→C
Operational Intensity [Flops/Byte]
APP-L1 (Cache-aware)
APP-L3 (Cache-aware)
APP-D (Original/Cache-aware)

APP-L1 vs. App-L3 vs. APP-D

- 정의: 같은 데이터셋을 N회 반복. 첫 반복만 DRAM 접근, 이후는 각 클래스의 주 접근 레벨로 동작.
  - APP-L1: 첫 반복 이후 주 접근 = L1
  - APP-L3: 첫 반복 이후 주 접근 = L3
  - APP-D: 모든 반복에서 DRAM 접근

Table: 1회 반복당 지표(β_0, φ_0, I)

클래스	주 접근 레벨 (반복 이후)	β_0 [bytes]	φ_0 [flops]	I (첫 반복)
APP-L1	L1	4.5 KB	72	1/64 ≈ 0.0156
APP-L3	L3	2 MB	524.3 K	1/4 = 0.25
APP-D	DRAM	61 MB	1023.4 M	16

Cache-Aware에서는 왜 I(작업강도)가 고정되나?

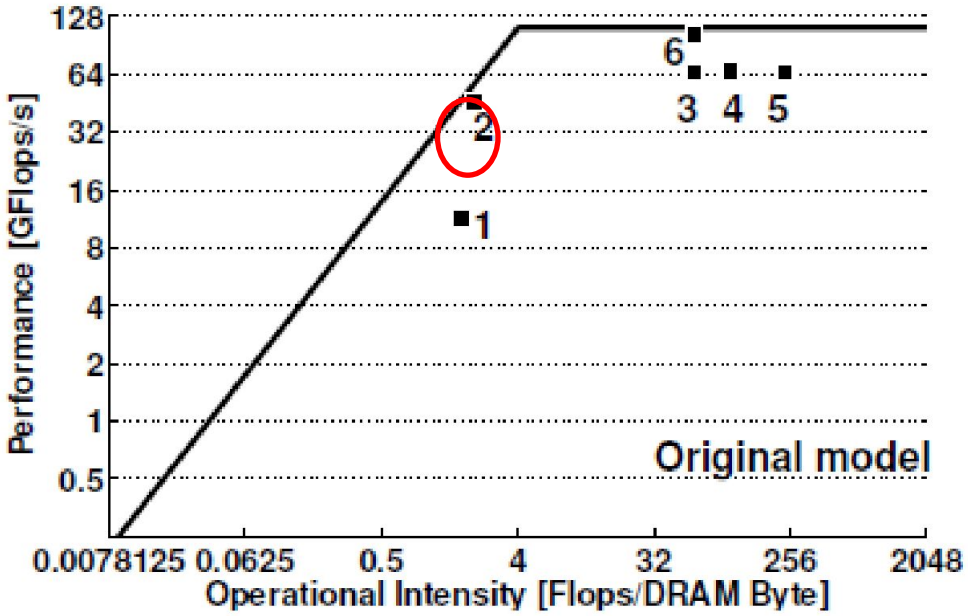
- 정의: $I = \varphi / \beta = (\varphi_0 \times N) / (\beta_0 \times N) = \varphi_0 / \beta_0 \rightarrow$ N이 소거되어 상수
- APP-L1, APP-L3, APP-D 모두 Cache-aware 정의에선 반복이 늘어도 같은 I를 유지
- Original DRAM- 중심 모델에서는 L1/L3의 경우 DRAM 바이트가 고정되어 I가 오른쪽으로 이동

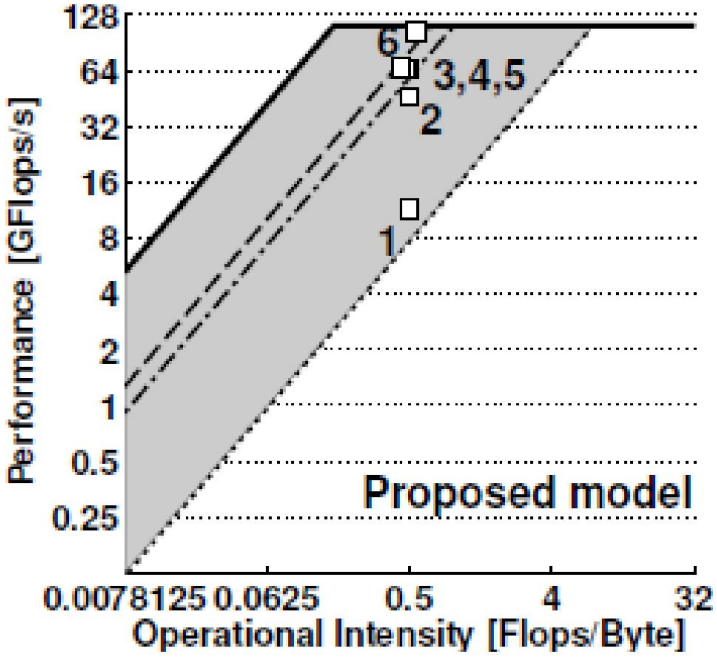
반복 N 증가에 따른 Flops/S 변화

- APP-L1: 첫 반복 DRAM 접근의 지연이 평균에서 희석 → L1→Core 대역폭 천장으로 수렴하며 Flops/s↑
- APP-L3: 첫 반복 후 L3 히트가 지배 → L3→Core 천장으로 수렴하며 Flops/s↑
- APP-D: 매 반복 DRAM 접근 → DRAM→Core 경로에 묶여 상대적으로 평탄

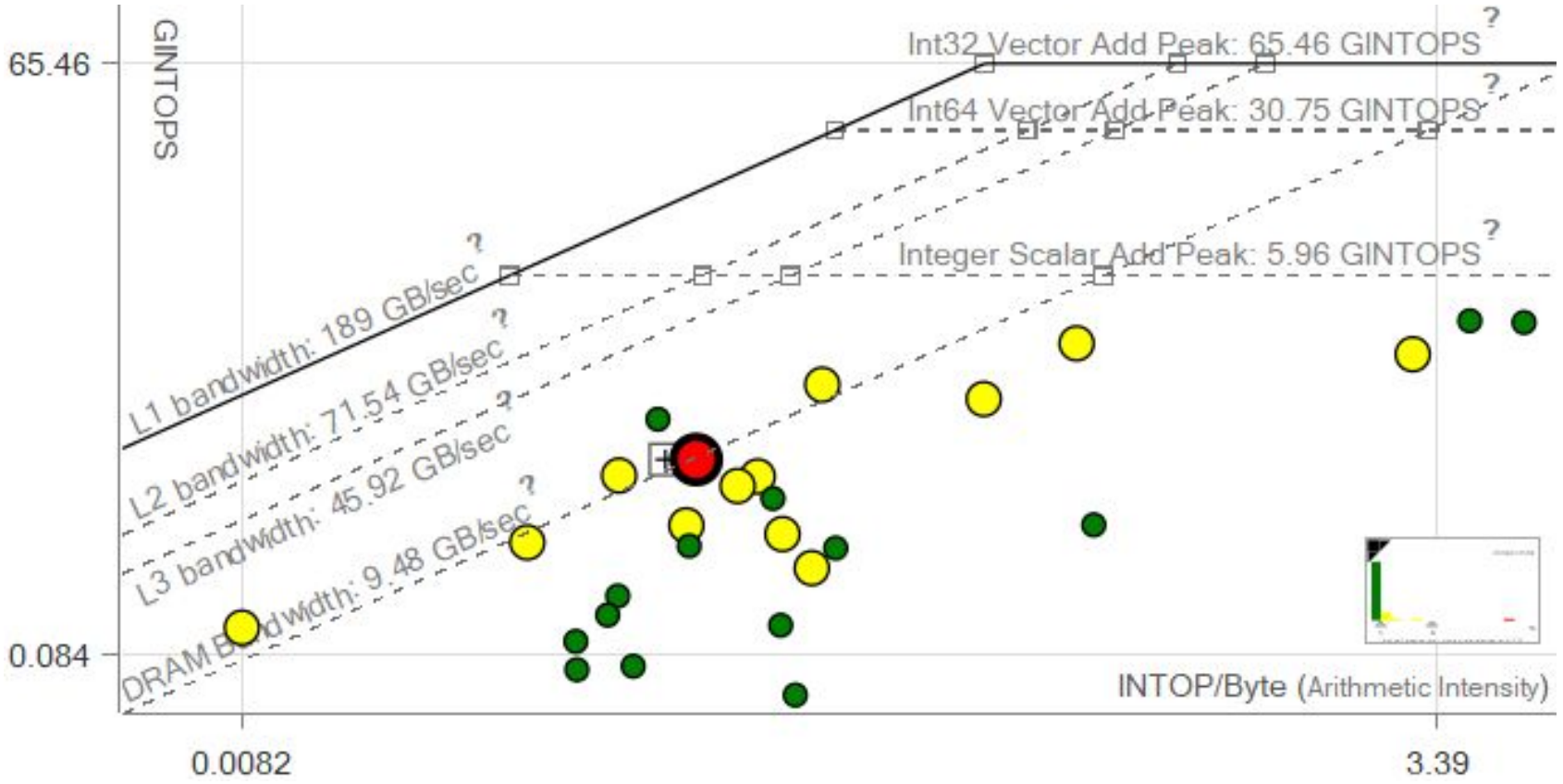
Cache-aware vs Original Roofline

- In the original roofline model, if an application is in the memory bound region, it is difficult to achieve the maximum attainable performance.



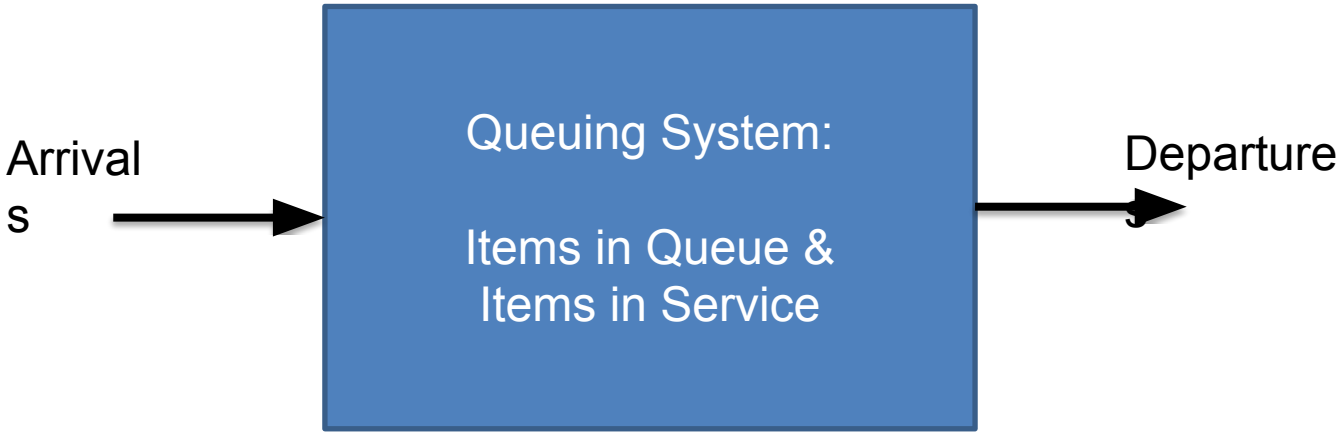


Cache-Aware Roofline Model in Intel Advisor



- Large red dot: takes up the most time,
  - candidate (loop or function) for optimization

Little's Law: **L = λ W**



L = average number of items in the queuing system
W = average waiting time in the system for an item (i.e. latency)
λ = average number of items arriving per unit time
(i.e., bandwidth)

Assumptions Stable, work-conserving system. Use long-run averages

Good Tool for Meeting Service-Level Objective (SLO) of a syster

Example 1A: 평균재학기간

- 매년 신입생이 2000명인 학교에서
  - λ = 2000/년 (avg throughput)
- 평균적으로 학교를 다니는 학생이 3000명이면
  - L = 3000

Q: 학생 1인당 학교에 다니는 기간은?

(avg latency) W = 3000/2000 = 1.5년

Design knobs via L = λW

Cap L: 동시 등록한 학생 수를 제한

Reduce W: 개설 강좌 증대, 선수과목 축소, 졸업학점 축소

Bound λ: 신입생 수 제한

Example 1B: 평균응답시간

- 매일 평균 50개의 이메일을 받는 사람이
  - λ = 50/day (avg throughput)
- 평균적으로 InBox에 150개의 응답하지 않은 이메일을 가지고 있다면
  - L = 150

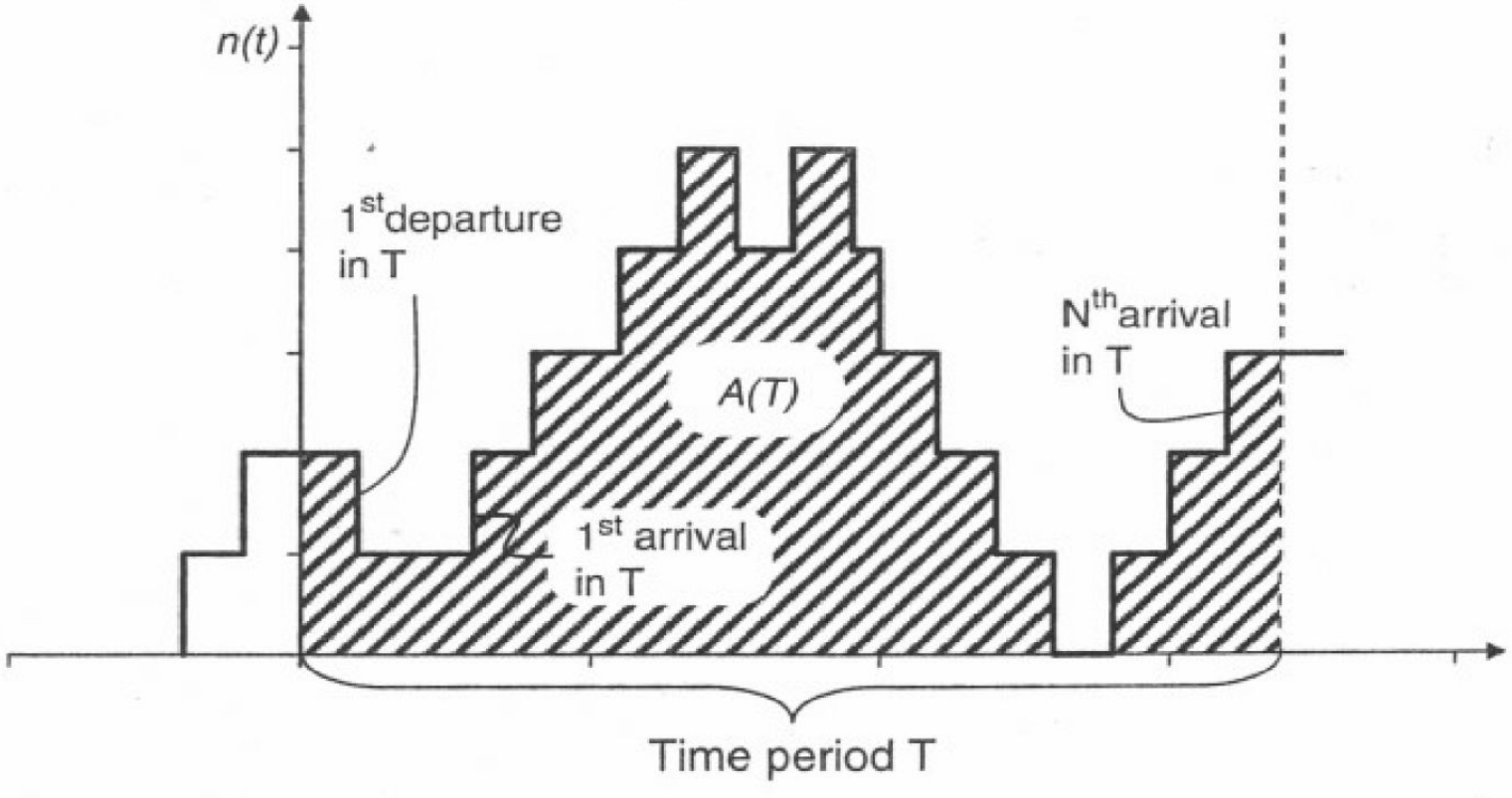
Q: 평균응답시간은?

- W = 150/50 = 3일

Design knobs via L = λW

Cap L: Set the max. inbox size
Reduce W: faster replies using templates
Bound λ: rate-limiting policy (office hours only)

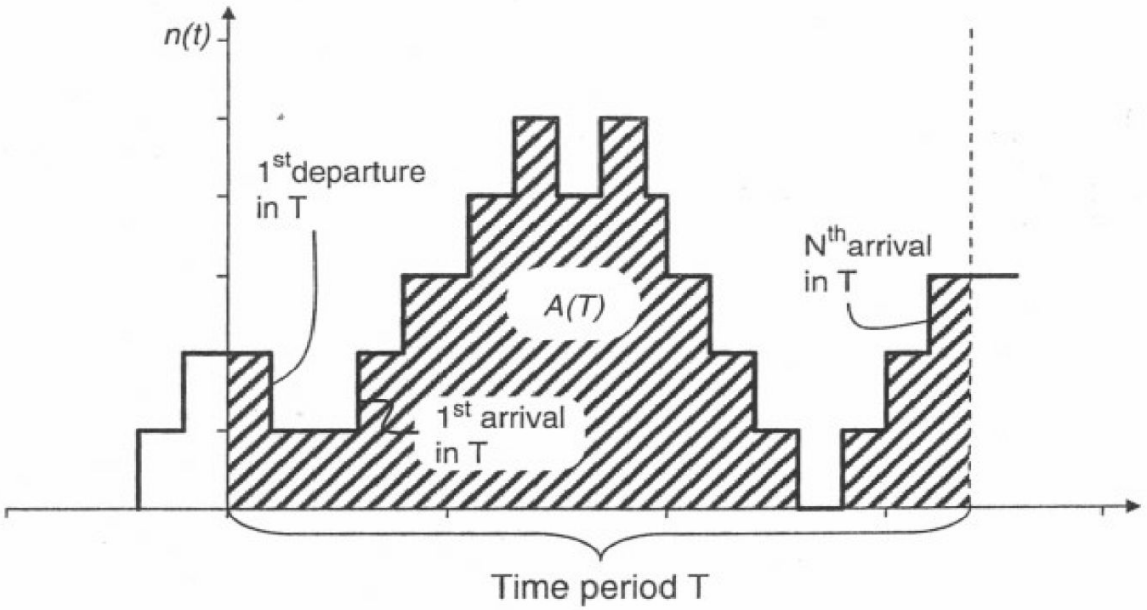
Intuitive Argument



n(t) = the number of items in queue at time t
T = a long period of time
A(T) = the area under n(t) over T
N(T) = the number of arrivals in T

- $\lambda(T) = N(T)/T$
  $L(T) = A(T)/T$
  $W(T) = A(T)/N(T)$

$\rightarrow L(T) = \lambda(T)W(T)$



$$\lim_{T\to\infty} L(T) = L$$
$$\lim_{T\to\infty} \lambda(T) = \lambda$$
$$\lim_{T\to\infty} W(T) = W$$

Example 2: Disk Utilization

For a single disk,

every second, 50 I/O requests are coming in &
avg disk service time = 10 ms,
what is the disk utilization?

답: 50 reqs/s x 0.01 s/req
= 0.50 (utilization)

- For a **stable, work-conserving single server** with **no losses**, the **utilization** (busy fraction) satisfies
  - $\rho = \lambda S$,

(cf) Little's Law:

- If W <= 0.1 s, $L_{max} = \lambda \times W_{max}$ = 50 reqs/s x 0.1 s = 5 reqs

Example 3: Buffer Requirements

Consider a cache memory with
the cache miss ratio = 6.25% &
the avg miss latency = 100 ns.

Q: If this cache receives 2 memory references per cycle at 2.5 GHz, how many buffers are needed for recording outstanding misses?

답: avg throughput =
2 refs/cycle x 2.5G cycles/s x 0.0625 misses/ref
= 0.3125 Gmisses/s

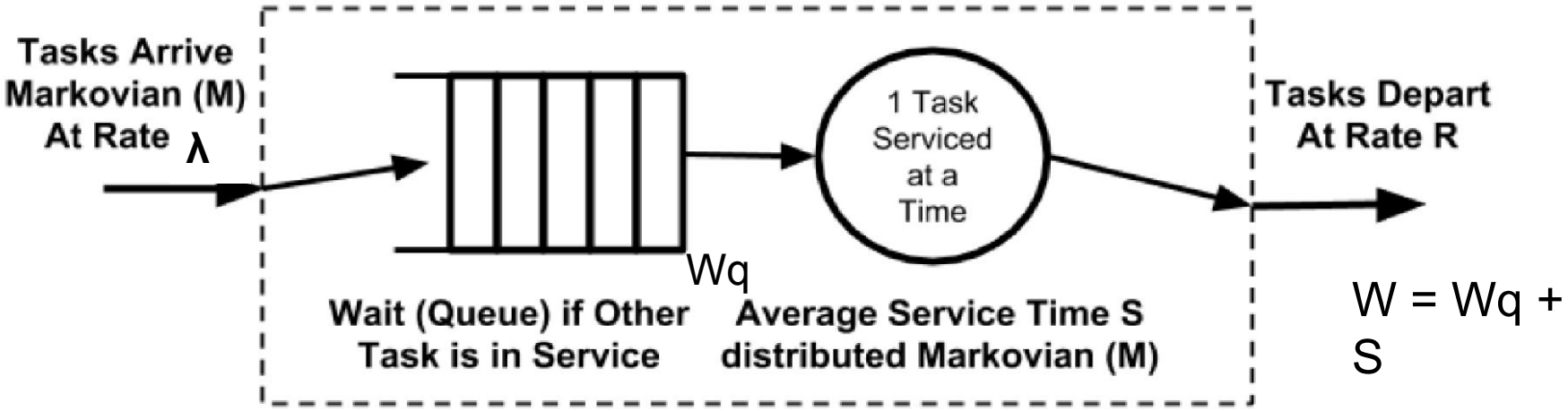
L = 0.3125 Gmisses/s x 100 ns = 31.25 misses

M/M/1 Queue

M: exponential interarrival times with mean 1/ λ

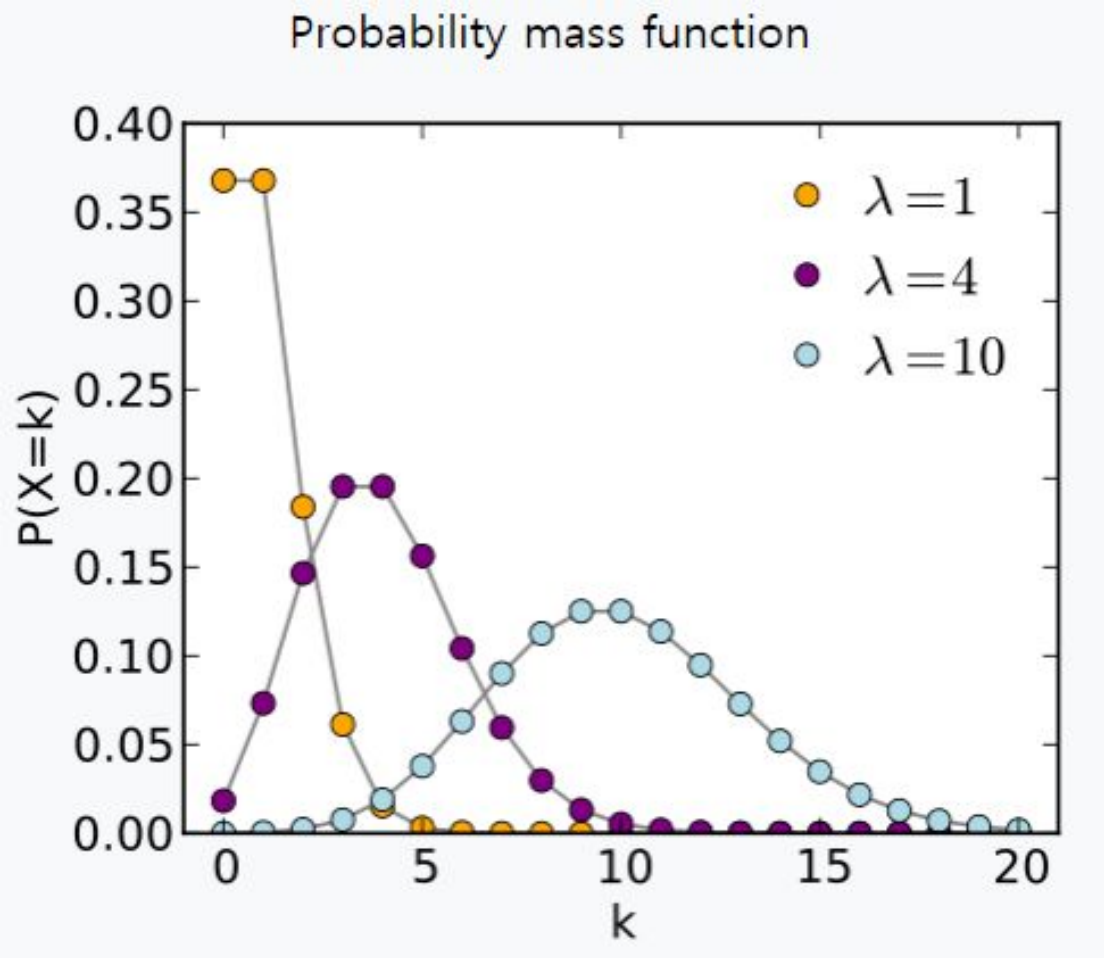
M: exponential service times with mean S

1: Single server



M/M/1 queue will reveal trade-off among (a) allowing unscheduled task arrivals, (b) minimizing latency , and (c) maximizing throughput

Poisson Distribution

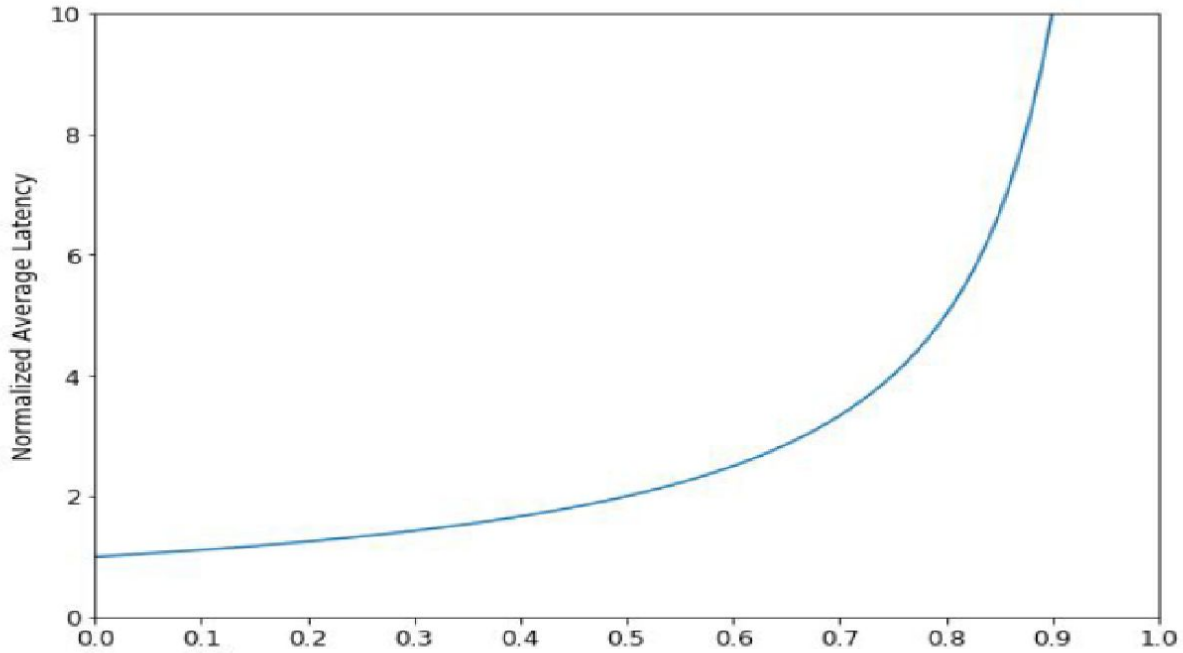


k: the number of occurrences

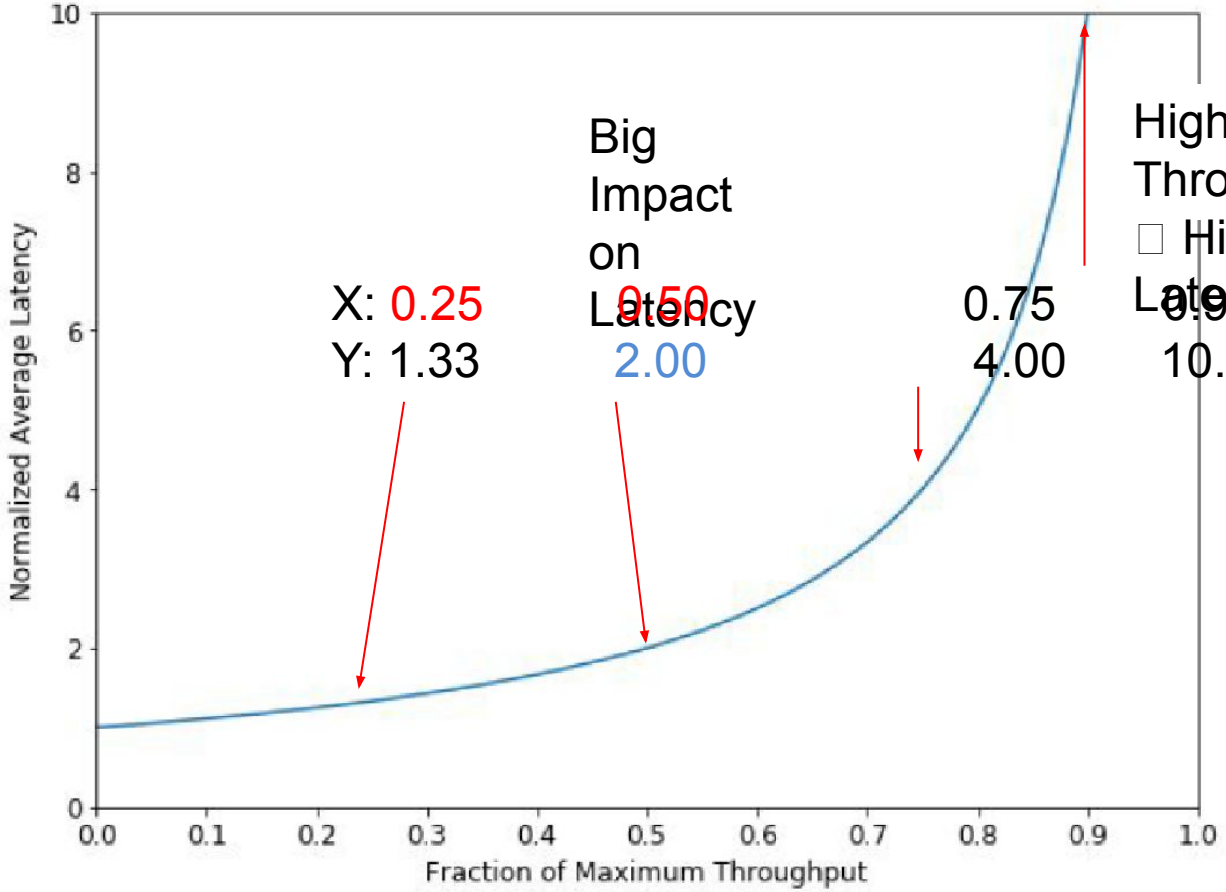
λ: the expected rate of occurrences

[Wikipedia]

M/M/1 Queue Latency vs. Throughput Tradeoff



From queuing theory, $y = 1/(1-x)$
where $y = W/S$ and $x = \lambda S$ (utilization)



Low Throughput
Low Latency

Example: M/M/1

Q: When λ = 10 packets/s, if we want to keep W = 100 ms, what S should be?

- Trial 1: Since $1/\lambda$ = 100 ms/packet, how about S = 100 ms??
- In M/M/1, S = 50 ms with Q = 50 ms

(1) Solve for S in $W/S = 1/(1- \lambda xS)$

$1/10S = 1/(1 – 10S)$

$1 – 10S = 10S$

$S = 1/20$ sec (λ x S = 0.5 □ 50% of max throughput)

(2) Solve for S in $W = S/(1- \lambda S)$

$1/10 = S/(1 – 10S)$

$10S = 1 - 10S$

$S = 1/20$ sec

Summary: Little's Law

- $L = \lambda \times W$ (items = throughput × latency)
  - Assumptions:
    - stable, work-conserving; use long-run averages
- Design levers via $L = \lambda W$
  - Cap L (WIP/concurrency)
  - Reduce W (service/overlap)
  - Bound λ (admission)

Use Case: From SLO to Limits

- Given SLO W_{SLO} and expected λ,
  - set $L_{max} = \lambda \times W_{SLO}$
  - Enforce with max concurrent requests/tokens, queue depth
  - If L_{budget} is fixed, $\lambda_{max} = L_{budget} / W$
  - Target utilization budget ρ between 0.6 and 0.8

Example: LLM Inference Serving

- Inputs: $\lambda = 100$ req/s, target $W_{SLO} = 150$ ms
  - $L_{max} = 100 \times 0.15 = 15$
    $\Rightarrow$ cap concurrent sequences ≈ 15
  - If current $W = 120$ ms at $\lambda = 100$
    $\Rightarrow L \approx 12$ (headroom ~3)
  - Operate near $\rho \approx 0.6$–0.8
    by tuning admission

Example: NVMe Queue Depth for Checkpoints

- Checkpoint 20 GB every 120 s ⇒ $\lambda \approx 171$ MB/s
  - Measured flush latency $W \approx 400$ ms ⇒ $L = \lambda W \approx 68$ MB in flight
  - With 1 MB I/Os ⇒ QD ≈ 68 (tune to p99, cap when saturation reached)
  - Re-evaluate if device S changes (e.g., thermal throttling, GC)

M/M/1: Essentials & the Knee

- λ arrivals, S = 1/μ service time, ρ = λS < 1
  - W = W_q + S (i.e., waiting + service)
  - W = S/(1−ρ), W_q = ρS/(1−ρ), L = ρ/(1−ρ)
  - Rule-of-thumb:

    plan for ρ ≈ 0.6–0.8 to avoid latency blow-up
  - Check with L = λW to keep units/budgets consistent

References

- M. Hill, “Three Other Models of Computer System Performance,” 2019. Available at https://arxiv.org/pdf/1901.02926.pdf.
- S. Williams et al.,“Roofline: An Insightful Visual Performance Model for Multicore Architectures,” CACM, April 2009.
- A. Ilic et al., “Cache-Aware Roofline Model: Upgrading the Loft,” IEEE CAL, 2014.
- A. Mohan, “Understanding Roofline Charts,” http://telesens.co/2018/07/26/....