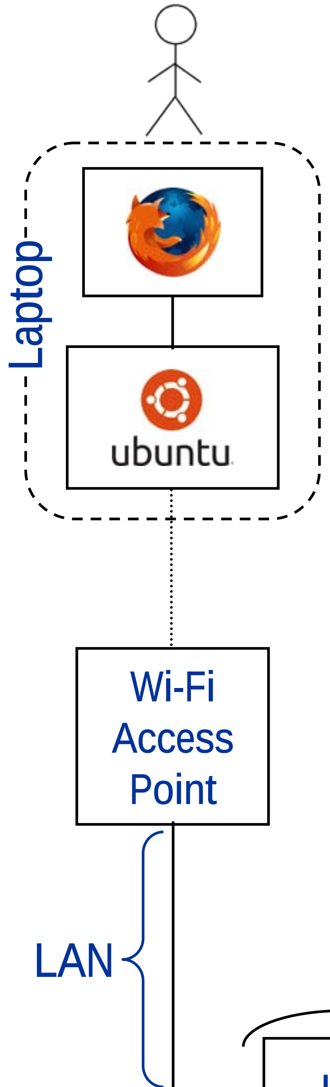
Why Are Distributed Systems Slow?

End to End RPC Latency



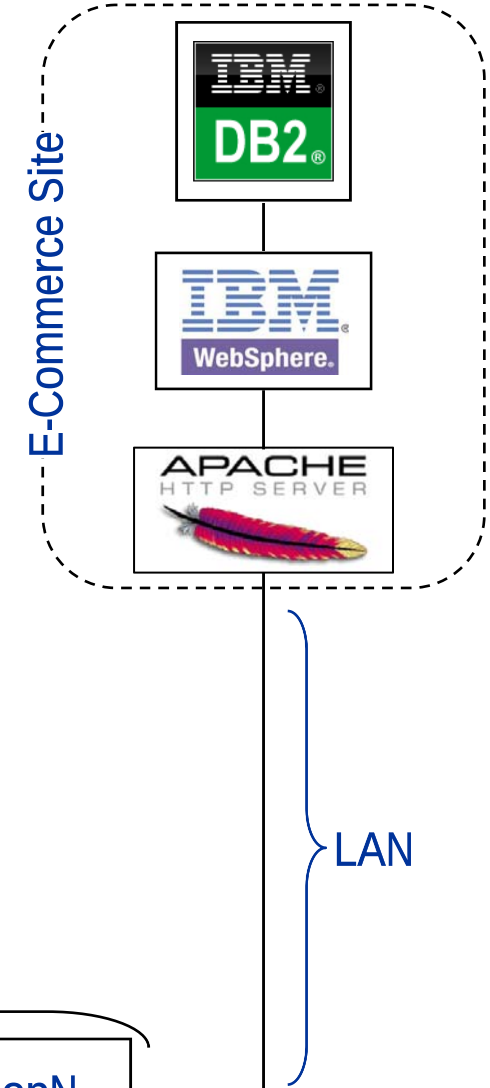
Follow the path of one RPC

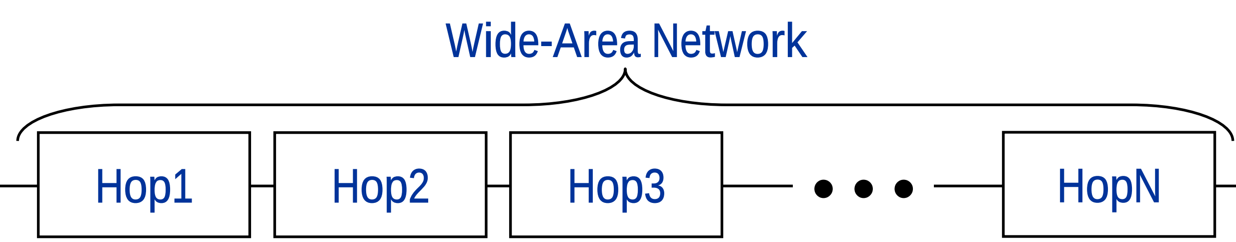
- request from Firefox to DB2
- reply back from DB2

Every step adds

1. Processing delay
2. Queueing delay
3. Transmission delay

Queueing delay often dominates





Queueing Theory 101

1. **Efficiency** **(i.e. resource utilization: e.g. CPU, network, disk, etc.)**
2. **Crispness** **(i.e., response time: user perception of quality)**
3. **Freedom** **(no advance reservations; just use when needed)**

You can have at most 2 out of 3

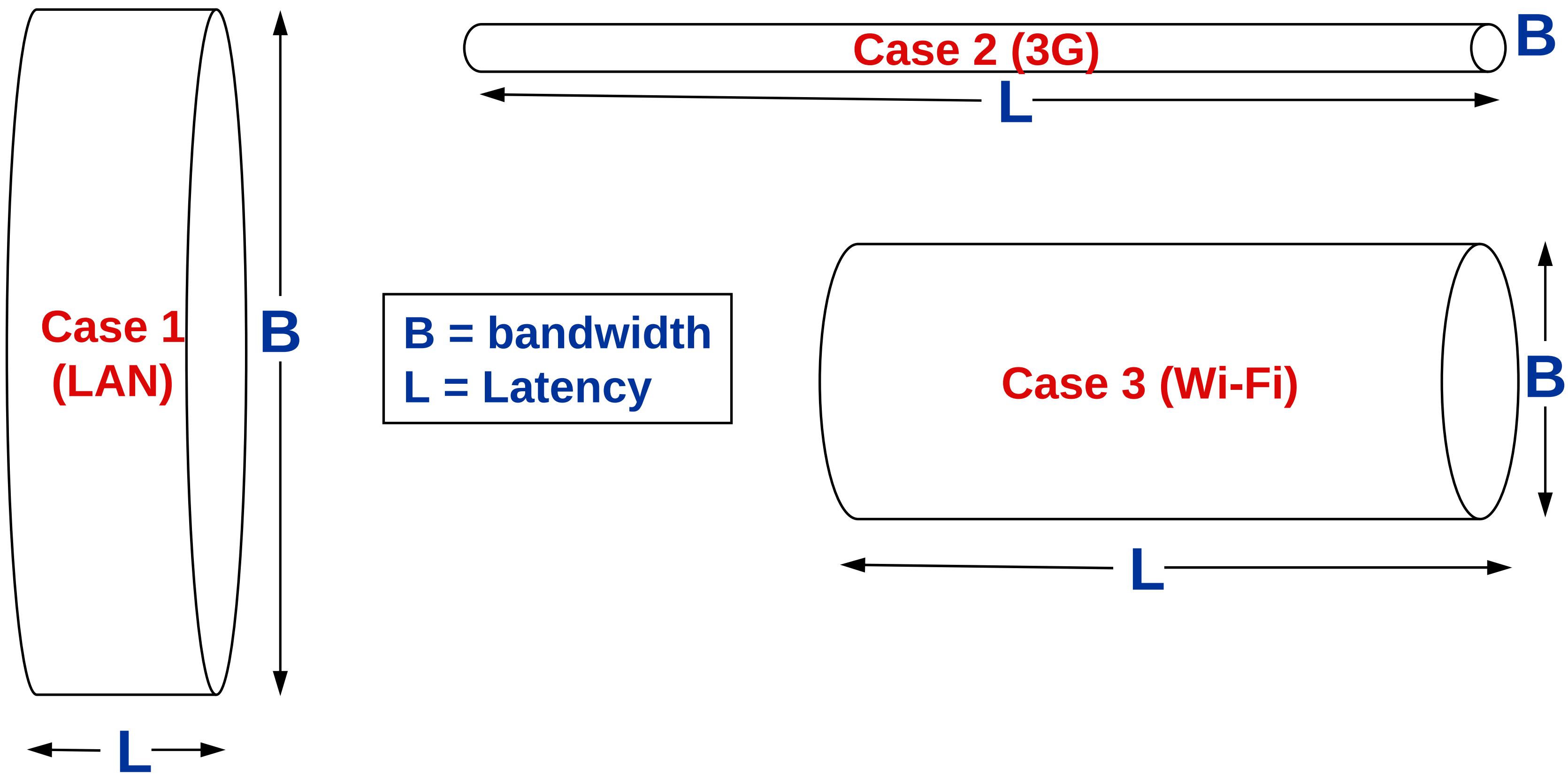
you can't have all 3 – no amount of cleverness helps

best you can achieve is satisfactory tradeoff

Alas

- **bean counters demand high efficiency (i.e. high utilization)**
- **freedom is non-negotiable**
- **so, crispness falls victim → long queues and high queueing delays**

Latency and Bandwidth



Delay-Bandwidth product ≈ max number of bits in flight

Latency

Latency is the killer, not bandwidth

- **fundamentally more difficult to improve**

Bandwidth improved through parallelism

- **fatter pipes, more lanes on highway, more checkout clerks at store, …**
- **does cost money, but not fundamentally difficult**

Latency is much, much, much harder to improve

- **typically requires deep structural changes**
- **e.g shorten distance, increase max speed tolerated, reduce path length**

Many software and systems trends increase latency

- **increased use of layers (middleware, external libraries (DLLs), VMs, …)**
- **security technologies (firewalls, overlay networks, …)**

Impact of Long Latency

Synchronous model of RPC becomes infeasible

- *coast-to-coast US $\approx$ 16 ms at speed of light*
- round-trip RPC > 30 ms
- larger distances (e.g. trans-Pacific) will make matters worse

30 ms is a long time in terms of lost processing opportunity

- 3 million instructions on an early 1990s processor (100 MIPS)
- much higher on faster processors and multi-core machines
- a modern 3GHz single-core x86 processor is ~1000-1500 MIPS
- so 30 ms is ~10-15 million instructions (or more, with multicore)

Can't afford to hide real-world asynchrony

- RPC is a synchronous abstraction
- inadequate for many emerging use cases
- more complex, asynchronous models needed
- more difficult to program and get correct

“Trust But Verify”

aka Optimistic Methods

Consider a very high latency distributed system

- **e.g. Mars rover has ~8 minutes RTT for tele-operation**
  (8m minimum, 48m maximum; depends on Earth-Mars distance)
- **RPC model simply won’t work; too slow and unresponsive**
- **only an optimistic method, with verification has hope (asynchronous)**

Information at one end about the other is always 8 min stale

latency-imposed limit, in this case speed of light

Approach: include a predicate to validate before execution

- **“When we last heard from you here was the situation…”**
- **“If this is still true, please do the following …”**
- **“If not, just tell us what happened and we will try to give you guidance”**

Even more extreme:

- **transatlantic communication before invention of telegraph**
  - **age of sail**
- **one-way latency was ~1 week; RTT was ~2 weeks**
- **how was command and control done?**
  - **e.g. King George and Lord Cornwallis during the American Revolution?**

For deeper discussion see optional reading:

"Fundamental Challenges in Mobile Computing"

Satyanarayanan, M.,

Proceedings of the Principles of Distributed Computing, 1996

"Trust but Verify" is an optimistic approach

- **contrast with simpler *pessimistic* approaches (lock or lease)**
- **more important as network latency dominates processing speed**
- **favors *liveness*, with controlled relaxation of *safety***