"If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."

— Sun Tzu, The Art of War

Today: two views on where the enemy lays when building [distributed] systems

[1]: Starting from a single system mindset

- Peter Deutsch's "8 fallacies"

[2]: Managing complexity: is the key difficulty in (computer) system building

Developing distributed systems: Pitfalls

[aka Peter Deutsch's "8 fallacies", cca'95] [1 [1, 2]

- The network
- Latency is ze
- Bandwidth is
- Tran
- The
- The
- The
- The network is

survey, 3, 4]

Note th

The Joys of Real Hardware

Typical first year for a new cluster:

~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)

Your network link: 10Gbps, 100ms RTT, 1KB MSS
Transport protocol: TCP

How long it will take to send an 100KB message?
(a 2x approx. factor is fine)

0.1ms
1.0ms
10.0ms
100.0ms
1,000.ms

Today: two views on where the enemy lays when building [distributed] systems

[1]: Starting from a single system mindset

- Peter Deutsch's "8 fallacies"

[2]: Managing complexity: is the key difficulty …

- *What are the external indicators of a complex system?*
- *What issues may come up?*
  - Emergent properties.
  - Propagations of effects.
  - [incommensurate] Scaling
  - Tradeoffs
- *What drives complexity?*
  - Large number of interacting features / requirements
  - Requirement for high performance / high utilization
- *What are your tools?*

Unlike constraints for engineering "real" systems:

Unconstrained design space

… no weight, physics laws, etc.

Key Challenge: **managing complexity** to efficiently produce *bug-free, maintainable, reliable*, *scalable*, *secure* [add your attribute here] software systems.

Slides dapted from: *Principles of Computer System Design: An Introduction*, J. Saltzer, F Kaashoek, Morgan Kaufman 2009.

Complexity

- Hard to define; symptoms:
  - Large # of components
  - Large # of connections
  - Irregular
  - No short description
  - Many people required to design/maintain

- Technology rarely the limit!
  - Indeed tech opportunity can be a problem
  - *The limit is usually designers' understanding*

Problem Types in Complex Systems

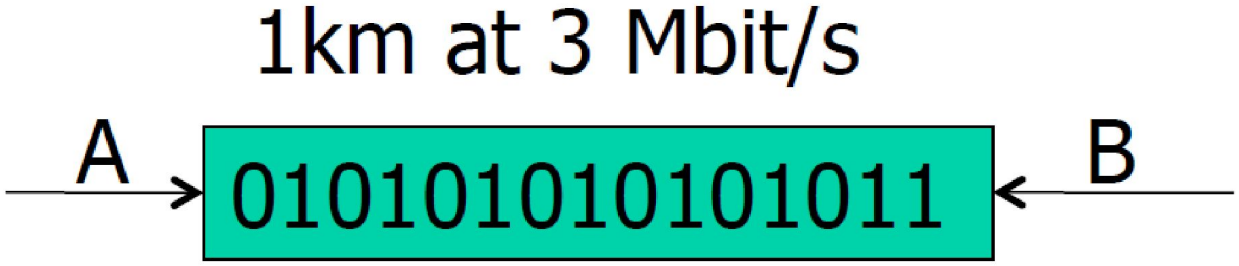
- Emergent properties
  - Surprises
- Propagation of effects
  - Small change -> BIG effect
- [Incommensurate] scaling
  - Design for small model may not scale
- Tradeoffs

Note: These problems will show up in non-computer systems as well

Emergent Property Example: Ethernet

- All computers share single cable
- Goal is reliable delivery
- Listen before send to avoid collisions

Will listen-while-send detect collisions?



- 1 km at 60% speed of light = 5 microseconds
  - A can send 15 bits before bit 1 arrives at B
- A must keep sending for 2 * 5 microseconds
  - To detect collision if B sends when first bit arrives
- **Minimum** packet size is 5* 2 * 3 = 30 bits

3 Mbit/s -> 10 Mbit/s

- Experimental Ethernet design: 3Mbit/s, 1km max segment
  - Default header is: 5 bytes = 40 bits
  - No problem with detecting collisions
- First Ethernet standard: 10 Mbit/s, 2.5km
  - Must send for 2*20 µseconds = 400 bits = 50 bytes
  - But Ethernet header is 14 bytes!
- Need to pad packets to at least 50 bytes
  - **Minimum packet size!**

Propagation of Effects Example (L. Cole 1969)

- **Attempt to control malaria in North Borneo …**
  - Sprayed villages with DDT
  - Wiped out mosquitoes, but ….
  - Roaches collected DDT in tissue
  - Lizards ate roaches and became slower
  - Easy target for cats
  - Cats didn't deal with DDT well and died
  - Forest rats moved into villages
  - Rats carried the bacillus for the plague
- **… Leads to replacing malaria with the plague**

Incommensurate scaling example (1)

Mouse -> Elephant (Haldane 1928)

- Mouse has a particular skeleton design
  - Can one scale it to something big? An elephant?
- Scaling mouse to size of an elephant
  - Weight ~ Volume ~ $O(n^3)$
  - Bone strength ~ cross section ~ $O(n^2)$
- Mouse design will collapse
  - Elephant needs different design!

Incommensurate scaling example (2)

- Scaling Ethernet's bit-rate
  - 10 Mbit/s: min packet 64 bytes, max cable 2.5 km
  - 100 Mbit/s: 64 bytes, (new) 0.25 km
  - 1,000 Mbit/s: (new) 512 bytes, 0.25 km

  - 10,000 Mbit/s □ design has collapsed!

    New design: no shared cable

Sources of Complexity

- Many goals/requirements
  - Interaction of features
- Requirements for Performance/High utilization

Example: More goals □ more complexity

1975 Unix kernel:

10,500 lines of code

2008 Linux 2.6.24 line counts:

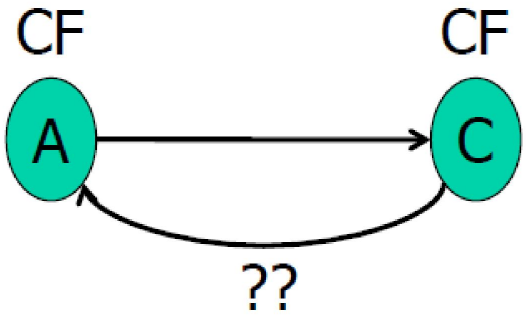
85,000 processes
430,000 sound drivers
490,000 network protocols
710,000 file systems
1,000,000 different CPU architectures
4,000,000 drivers
7,800,000 Total

Growth in size of Linux Kernel between 1991 and 2003

Example: Interacting features more complexity

Some simple features ..

- Call Forwarding



- Call Number Delivery Blocking
- Automatic Call Back
- Itemized Billing

- A calls B, B is busy
- Once B is done, B calls A
- A's number on appears on B's bill

…assemble to complex system-level behavior

Interacting Features

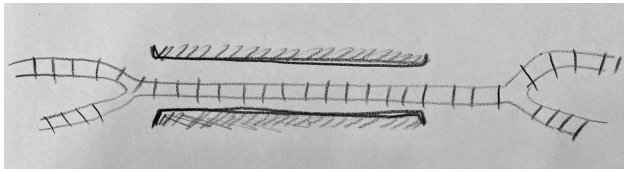
The point is not that these interactions can't be fixed …

- … but that the [many] interactions that have to be considered [and possibly treated as special cases] are a huge source of complexity.
- Possibly n^2 interactions or more,

- Cost of thinking about / fixing interaction gradually grows to dominate s/w costs.
- Complexity is super-linear (with # features)

Example: More performance -> more complexity

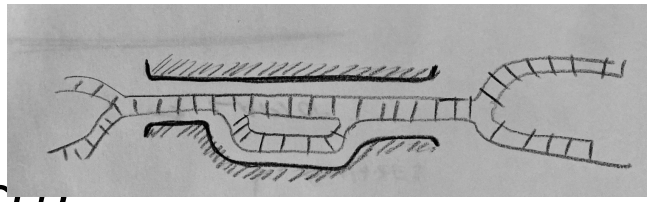
One rail track in a narrow canyon



- Base design: alternate trains
  - Low "performance": low throughput, high delay, low utilization
  - Worse than two-track, but cheaper than blasting

- Improved design? (increase utilization at low capital cost)

Increase utilization w/ a siding and two trains



  - Is a new signaling design needed?
  - Precise schedule
  - *Siding limits train length! (a global effect!)*

Cost of increased 'performance' is often supra-linear

Summary of examples

- Expect surprises!
- There is NO *small* change.
  - "Just one more feature!"
- 10x increase ⇒ perhaps re-design
- Performance cost is super-linear

Coping with Complexity

[a lens to view most CPEN courses]

Simplifying insights / experience / principles / successful designs

- E.g., "Avoid excessive ge

Modularity

- Split up system,

Abstraction

- Hide implementatio
  propagation of chan

Hierarchy

- Reduce connections

Layering

- Gradually build up capabilities
- Reduce connections

BugCount: ~ N (lines of code)

Debug time: ~ N x *BugCount*

~ N^2

---------- Split in K modules

Debug time: ~ K x $\left(\frac{N}{K}\right)^2$

~ $\frac{N^2}{K}$

Coping with Complexity

[a lens to view all CPEN courses: e.g., CPEN221]

Simplifying insights / experience / principles

- E.g., "Avoid excessive generality", **immutability, design patterns, idioms**

Modularity

- Split up system, consider small pieces separately
- **Data Types**

Abstraction

- Helps avoiding propagation of effects
- **Specifications: hide internals of implementation / hide usage details**
- **Abstract Data Types**

Hierarchy

- Reduce connections / regular layouts
- **Subtyping, Divide-and-conquer,**

Layering

- Gradually build up capabilities
- Reduce connections

Coping with Complexity

[This course: CPEN431]

Simplifying insights / experience / principles

- **Theoretical constructs: reasoning about time and event ordering, commit protocols, CAP theorem**
- **Analyze [successful/failed] designs and building blocks (e.g., end-to-end principle, soft-state, DHTs, epidemic protocols,**
- **Lots of hands-on experience, define tradeoff space; feeling for what matters**

Modularity and abstraction

- Split up system, consider parts separately
- Hide implementation internals / hide usage details
- **Technology for remote modules/abstraction: RPC, Protocol Buffers.**
- **Effects of decomposition with unreliable components/networks**

Hierarchy - Reduce connections / regular layouts

Layering

- Reduce connections. **Most designs in this space. Cross-layer optimizations.**

- A1: Server is up. Assignment is due Monday Jan 16th
- A2: Server is up. Assignment is due Friday Jan 20th
  - Take a look at FAQs (at the bottom of A1/A2 descriptions)
- A4: description is up (there is no A3).
  - May be useful to start and have an early prototype.

- A1: assignment is up. Due Monday Jan 16th
- A2: assignment is up. Due Friday Jan 20th
  - Same story as in A1 but:
    - Protocol Buffers rather than ad-hoc serialization protocol
    - Checksums
    - Opportunity to rethink you design to separate request/reply layer from application level layer
  - Server is up
  - Take a look at FAQs (at the bottom of A2 description)
- A3 and A4: descriptions are up.
  - May be useful to start on A3 and have an early prototype.