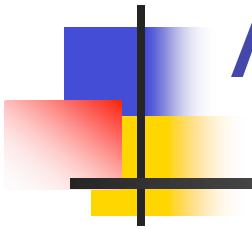
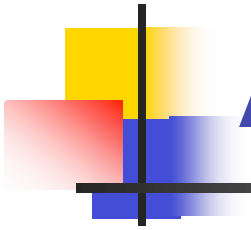


Architecting for Latency

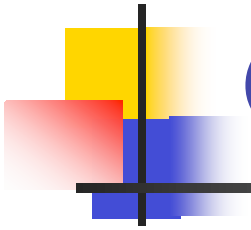
Dan Pritchett

eBay, Inc.



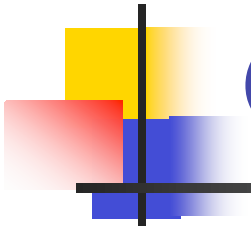
Agenda

- What causes latency?
- Why consider it during architecture?
- What are the challenges with latency?
- What are the solutions?



Geographic Realities

- Business Continuity (i.e. Disaster Recovery)
  - Best practices dictate diversity of
    - Geographies
    - Networks
    - Power
  - Continuity models
    - Active/Passive
    - Active/Active

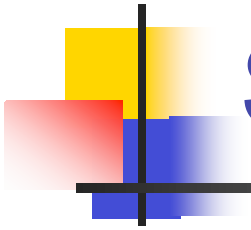


Global Markets

- Internet has created a global economy
  - Global trade overtaking domestic trade
  - Corresponding infrastructures also adapting (shipping, tariffs, etc.)
- Network latency from customers to services is a reality
  - Demand for distributed services growing
    - Shifts latency to architectures away from customers.

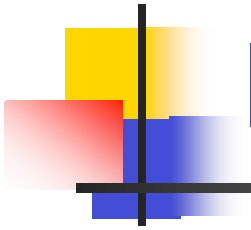
US Latencies

Keynote Data, August 8, 2007					
	ATL	BOS	MCI	NYC	SFO
ATL	3	36	33	24	76
BOS	33	2	43	9	74
MCI	35	52	2	39	49
NYC	26	9	40	3	77
SFO	78	74	48	78	2



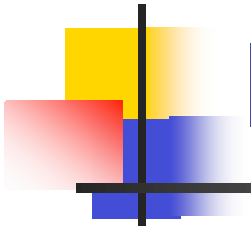
Service Latency

- Component A depends upon component B
  - Client A invokes Service B
    - A's response time is ≥
      - B's processing time +
      - Latency of path between A and B
  - Availability
    - System availability, product of
      - Availability of A
      - Availability of B



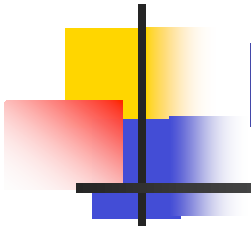
Impact of Latency

- Performance
  - Slower response times
- Resources
  - Synchronous designs
    - Increased thread and memory usage
  - Asynchronous designs
    - Storage for queues
    - Added processing



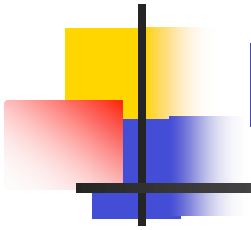
Irrational Thoughts

- Latency is the dark secret of architecture
- Often not well understood or even considered
- Which leads to the following irrational thoughts…



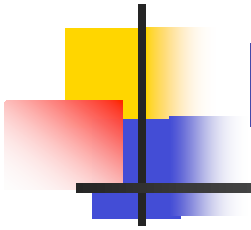
Irrational Thought #1

- Latency can be ignored
  - Corollary to Distributed Computing Fallacies #2 (Latency is zero)
  - Reality
    - Latency slows synchronous interactions
      - Worse case, latency exceeds processing
    - Latency consumes critical resources
      - Longer response times = more threads, more memory
      - Difficult to tune typical request/response architectures to cope with latency



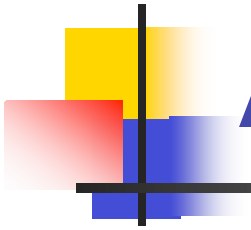
Irrational Thought #2

- Predictability is necessary
  - Latency introduces variability
  - Variability is the antithesis of predictability
  - Reality
    - Impossible to achieve predictability results from unpredictable inputs
    - Complexity unavoidable when ignoring axioms.



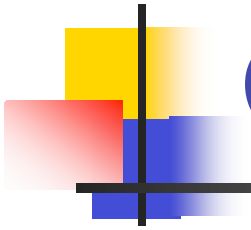
Irrational Thought #3

- Persistent state is always consistent
  - Globally consistent state is impractical and unnecessary
  - Reality
    - Multi-phase commits intolerant of latency
    - Forcing consistency limits alternatives



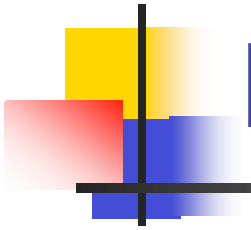
Architectural Tools

- Loose deployment coupling
  - Focus on deployment, as well as interfaces
- BASE
  - An alternative to ACID that scales across latent paths.



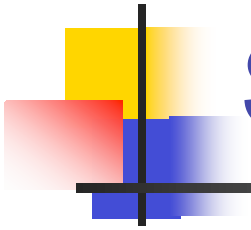
Coupling

- What is coupling?
  - Causing A to depend upon B in such a matter that changes to B forces changes to A
- Interface vs. Deployment
  - Interface defines functional couplings
  - Deployment defines the “ilities”
    - Performance, availability, latency



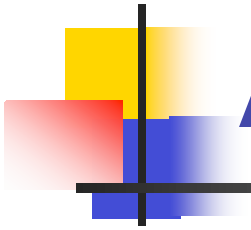
Deployment Decoupling

- Why worry about deployment coupling?
  - Topologies become constrained
    - Network topology becomes important
  - Hardware resources influence applications
    - Small soldier vs. big soldier
  - In general, deployment becomes brittle and non-scalable.



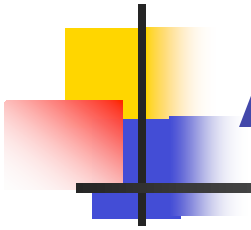
Synchronous Coupling

- Synchronous dependencies are tight deployment coupling
  - Availability
    - A is down if B is down
  - Performance
    - A is slow if B is slow
  - Scalability
    - B must grow if A grows



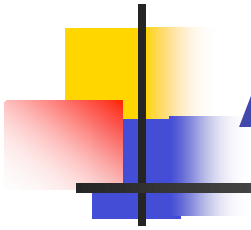
Asynchronous Decoupling

- What if A can message B?
  - A's availability is independent of B
    - Caveat: Queues for B will obviously grow if B is unavailable
  - A's performance is independent of B
  - A can scale independently of B
    - Caveat: B obviously must be able to manage arrival rate of A
      - But depending up on SLA's, B can use off-peak cycles to catch up.
      - More flexibility in scaling A and B independently.



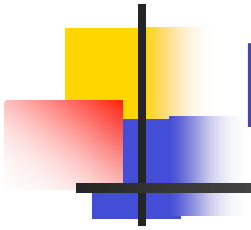
Asynchronous Candidates

- Prefer large to small components
  - Good
    - Full text search integration
    - Billing
    - Payments
  - Poor
    - Database access
- Ideal candidates are any interfaces that are primarily unidirectional.



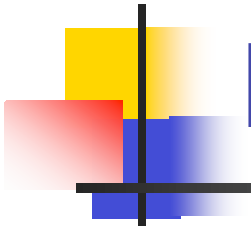
Asynchronous Integration

- Messaging Systems
  - Variety of options
    - Trade-off of:
      - Throughput
      - Latency
      - Reliability
- Event architectures
  - Similar to messaging



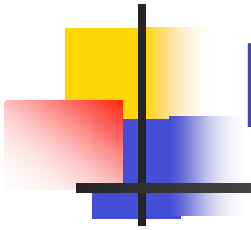
Messaging Features

- Some features expensive, but necessary?
  - Exactly once delivery
    - Is your application domain inherently idempotent?
    - Often less expensive in application domain than messaging platform
  - Ordered delivery
    - Dependencies between events is generally wrong
      - See Irrational Thought #2 (Predictability is necessary)



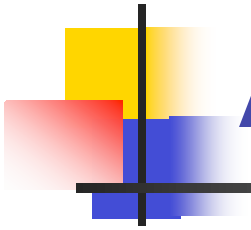
Event Architectures

- Event Stream Processing (ESP)
  - Event streams processed by a SQL like language
    - Events are rows, attributes are columns
    - Temporal and volume based sets
    - Query results can be data sets or new events
- Efficient approach for managing analysis of large data streams
  - And provides loose deployment coupling.



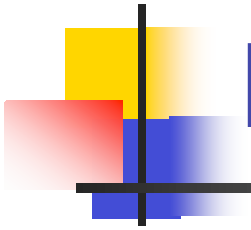
BASE

- A latency tolerant alternative to ACID
  - Basically Available
  - Soft state
  - Eventually consistent
- Derived from CAP Theorem
  - Pick two from below:
    - Consistency
    - Availability
    - Partitioning



ACID vs. BASE

- ACID
  - Strong consistency
  - Pessimistic
  - Focus on commit
  - Isolation
  - Difficult schema evolution
- BASE
  - Weak consistency
  - Optimistic
  - Focus on availability
  - Best effort
  - Flexible schema evolution
  - Approximate answers okay
  - Faster
  - Simpler

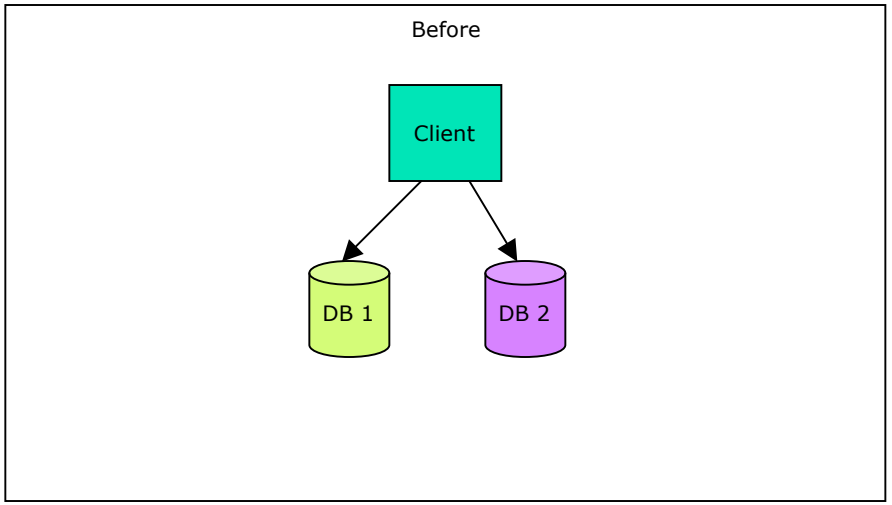


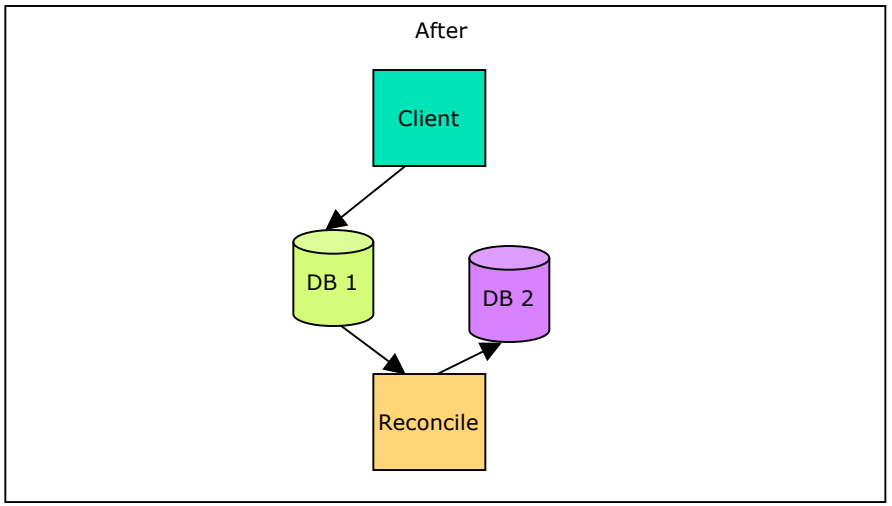
BASE and Latency

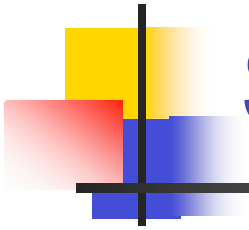
- Why does BASE help?
  - Free us of the irrational thoughts
    - Best effort is not predictable
    - Weak consistency is permitted
  - Pattern for partitioning
  - Inherent loose deployment coupling

ACID vs. BASE, Illustrated

- Before
  - 2PC commit to DB1 and 2
    - Client availability coupled to both
    - Latency on both paths critical
- After
  - Single commit to DB1
    - Client only dependent upon DB1
  - Reconcile asynchronously
    - Latency tolerant
    - Decoupled availability







Summary

- Latency is real
- Irrational thoughts lead to brittle architectures
- Tools for architects
  - Asynchronous Integrations
    - Messaging
    - ESP
  - BASE
    - White paper on BASE/CAP
      - http://citeseer.ist.psu.edu/544596.html