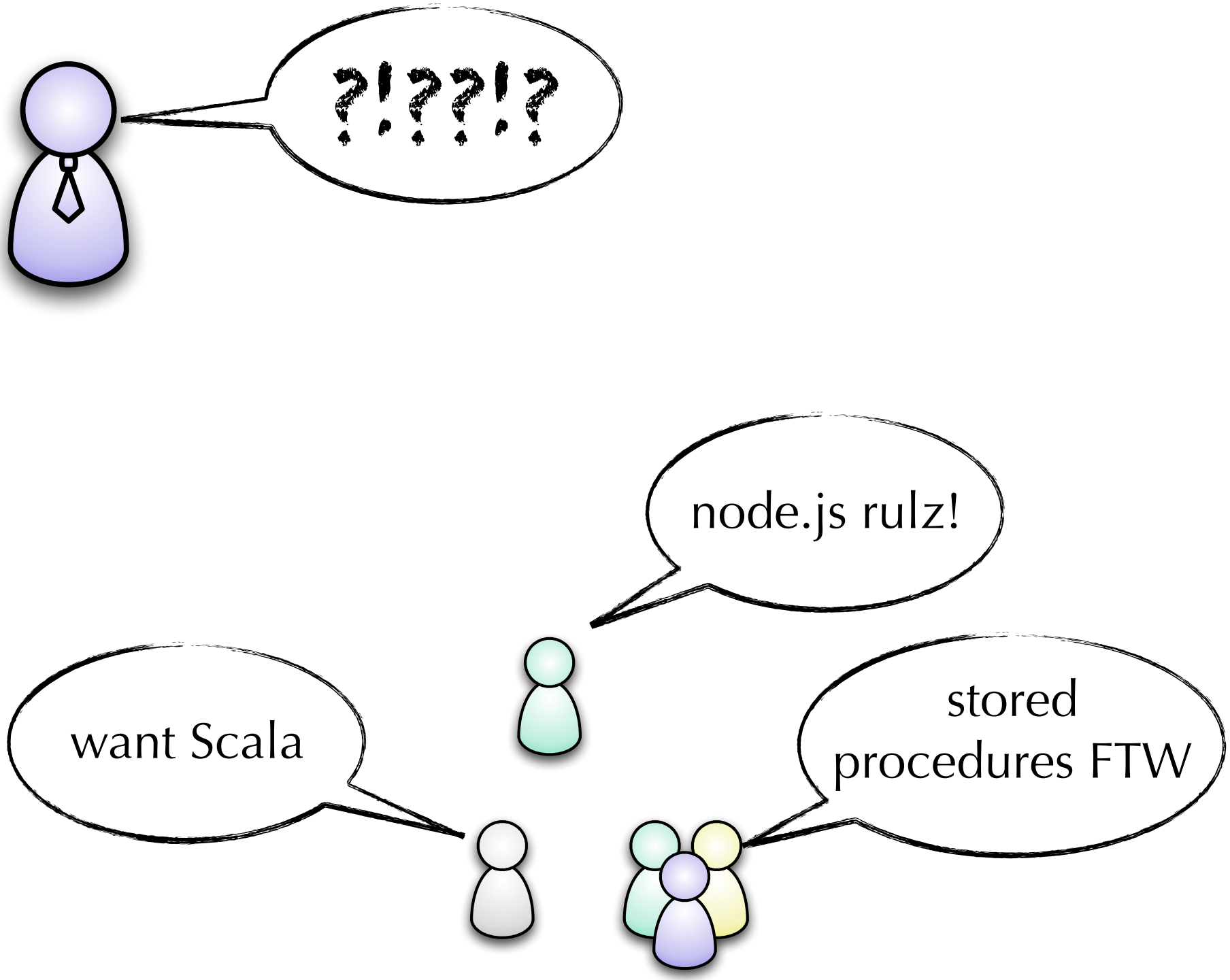
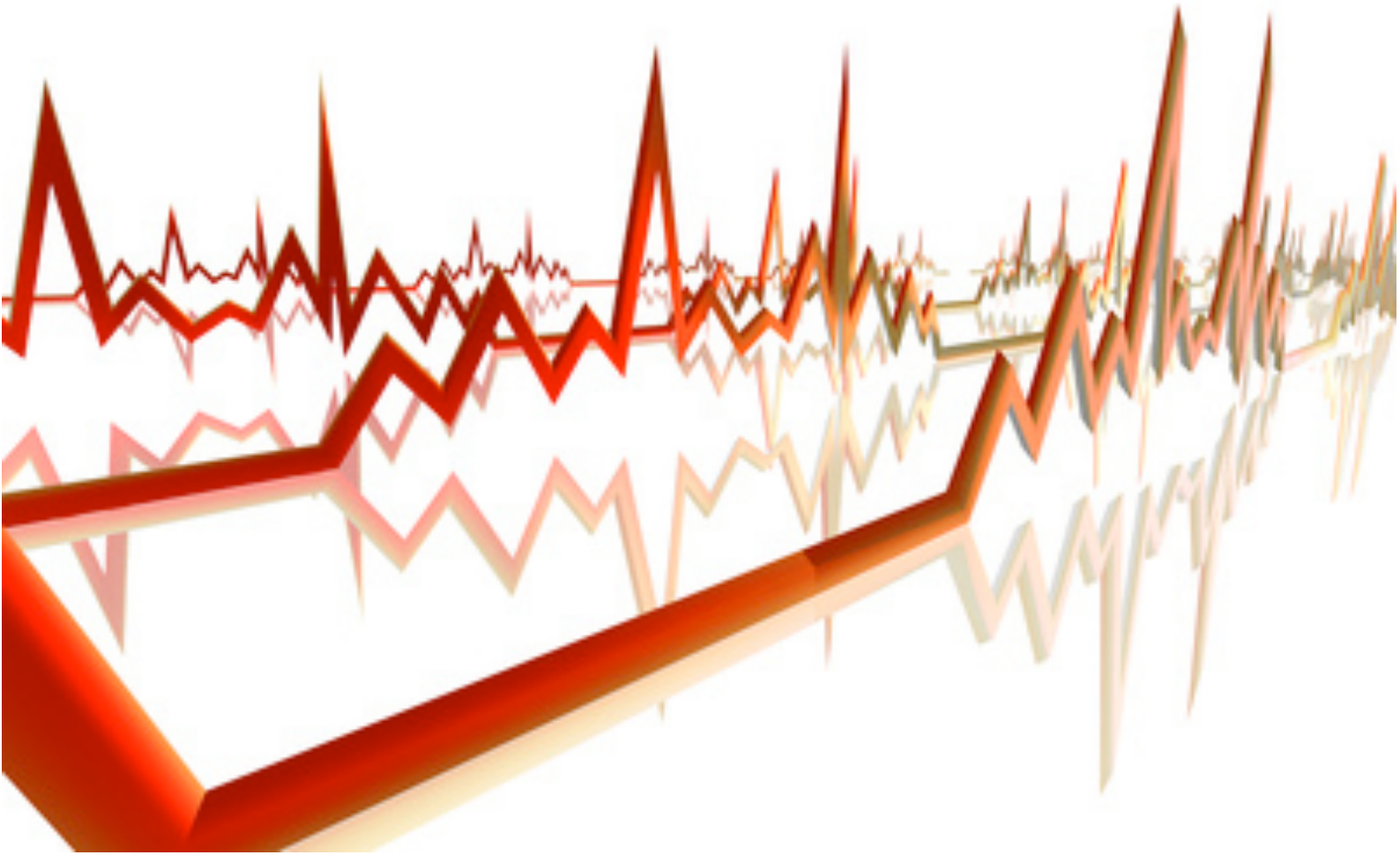
comparing architectures

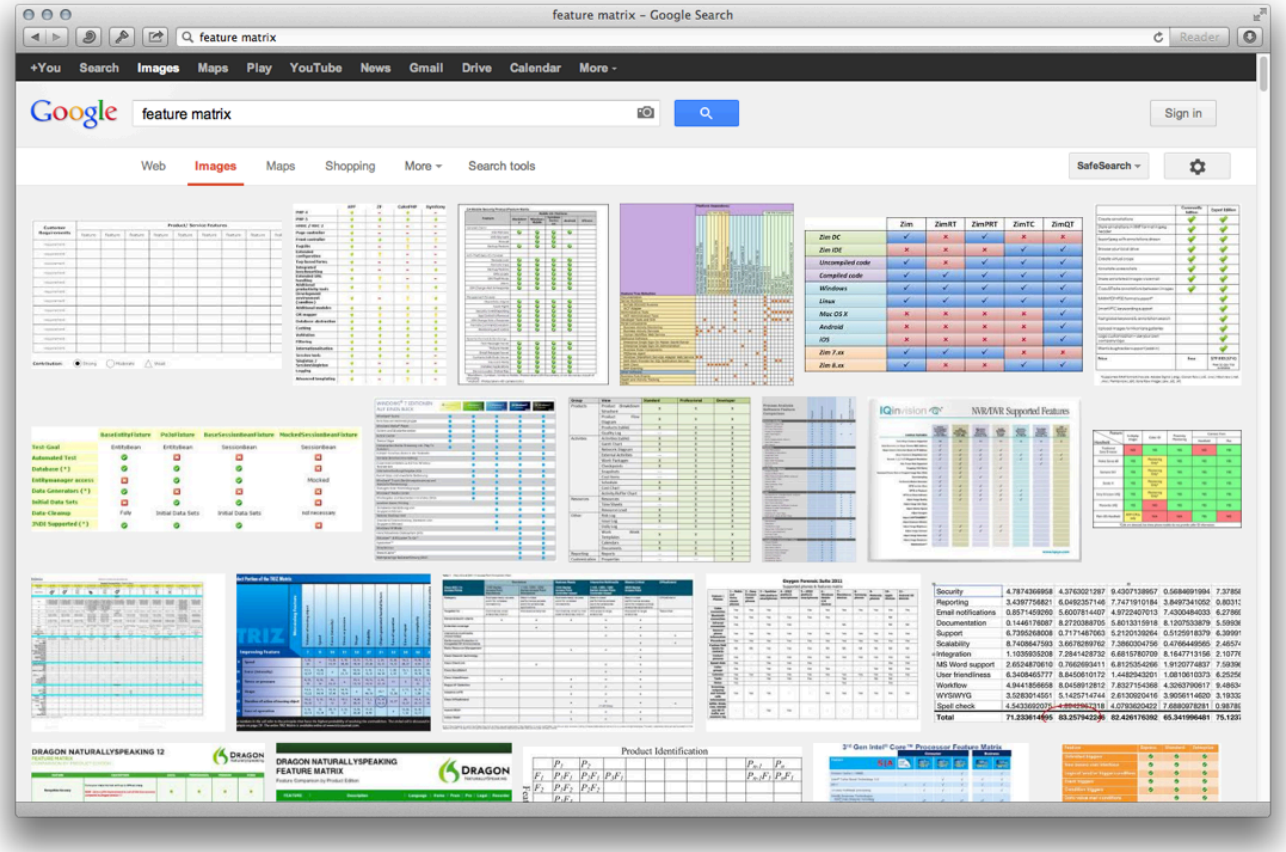

?!?!?!?
node.js rulz!
want Scala
stored
procedures FTW


choosing technology

understand current trends



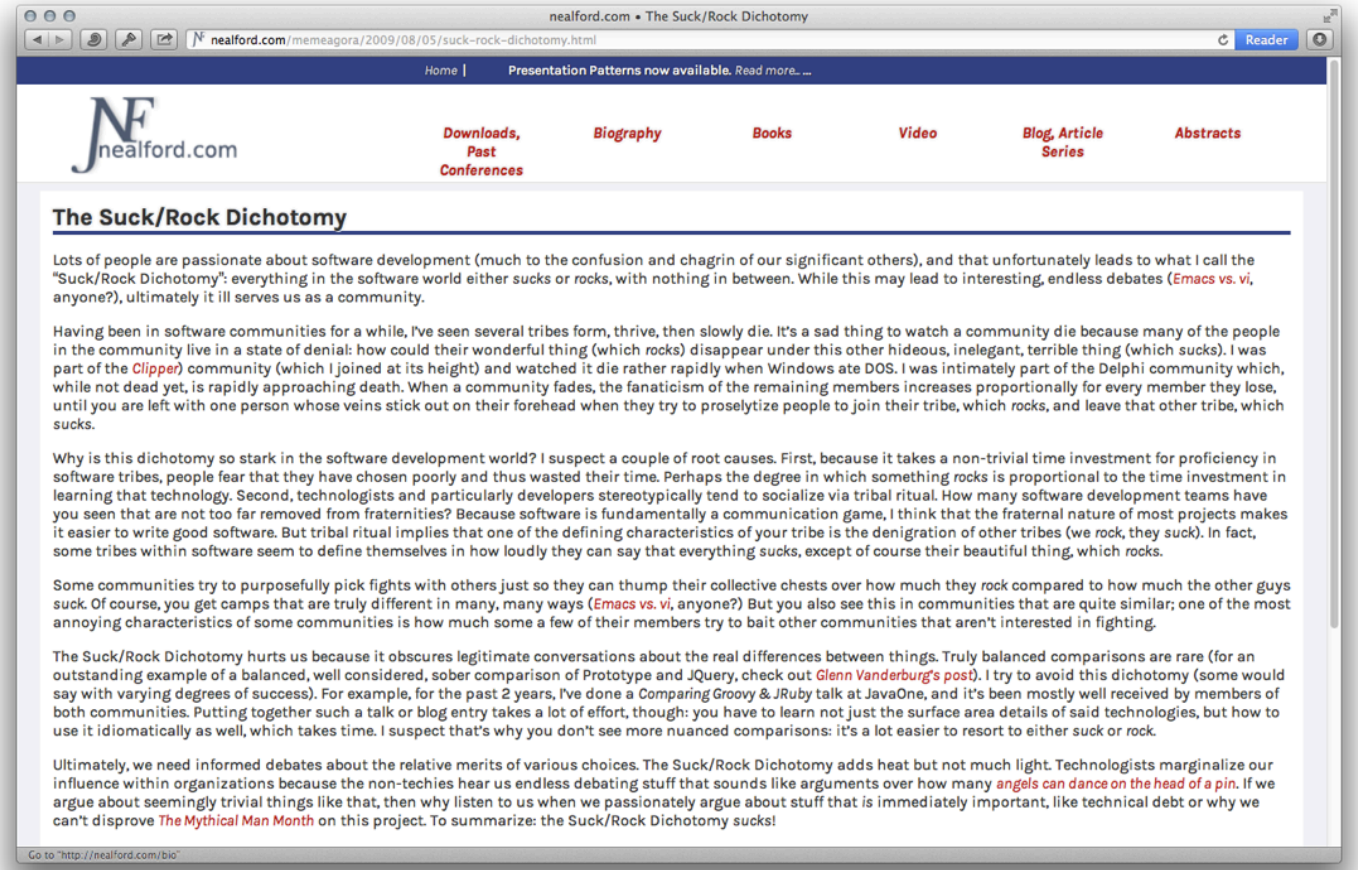
normalize feature sets



build your own feature matrix

good, bad, ugly

avoid the “suck/rock” dichotomy



nealford.com/memeagora/2009/08/05/suck-rock-dichotomy.html

spikes

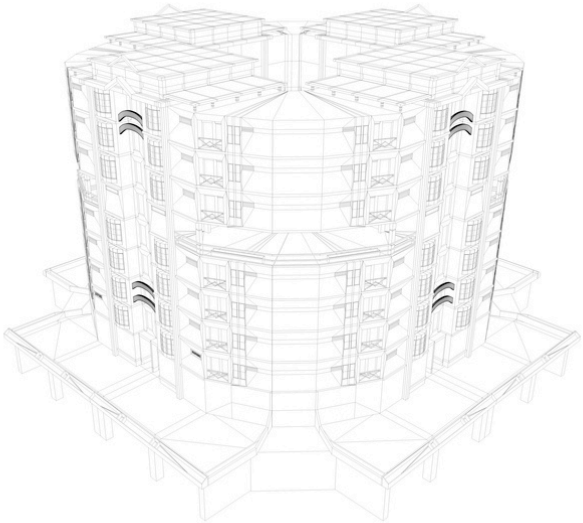
time-boxed experimental coding exercise

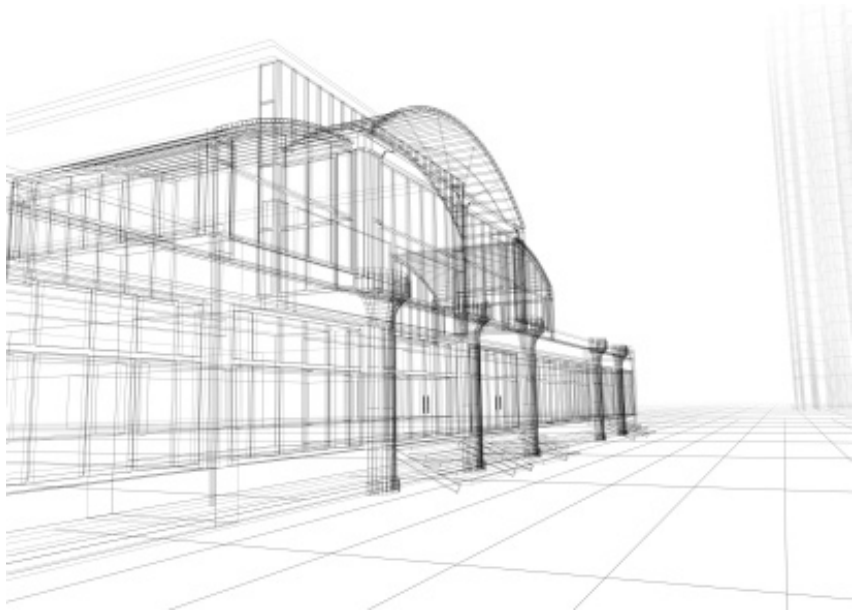
pure knowledge acquisition

NOT a prototype!

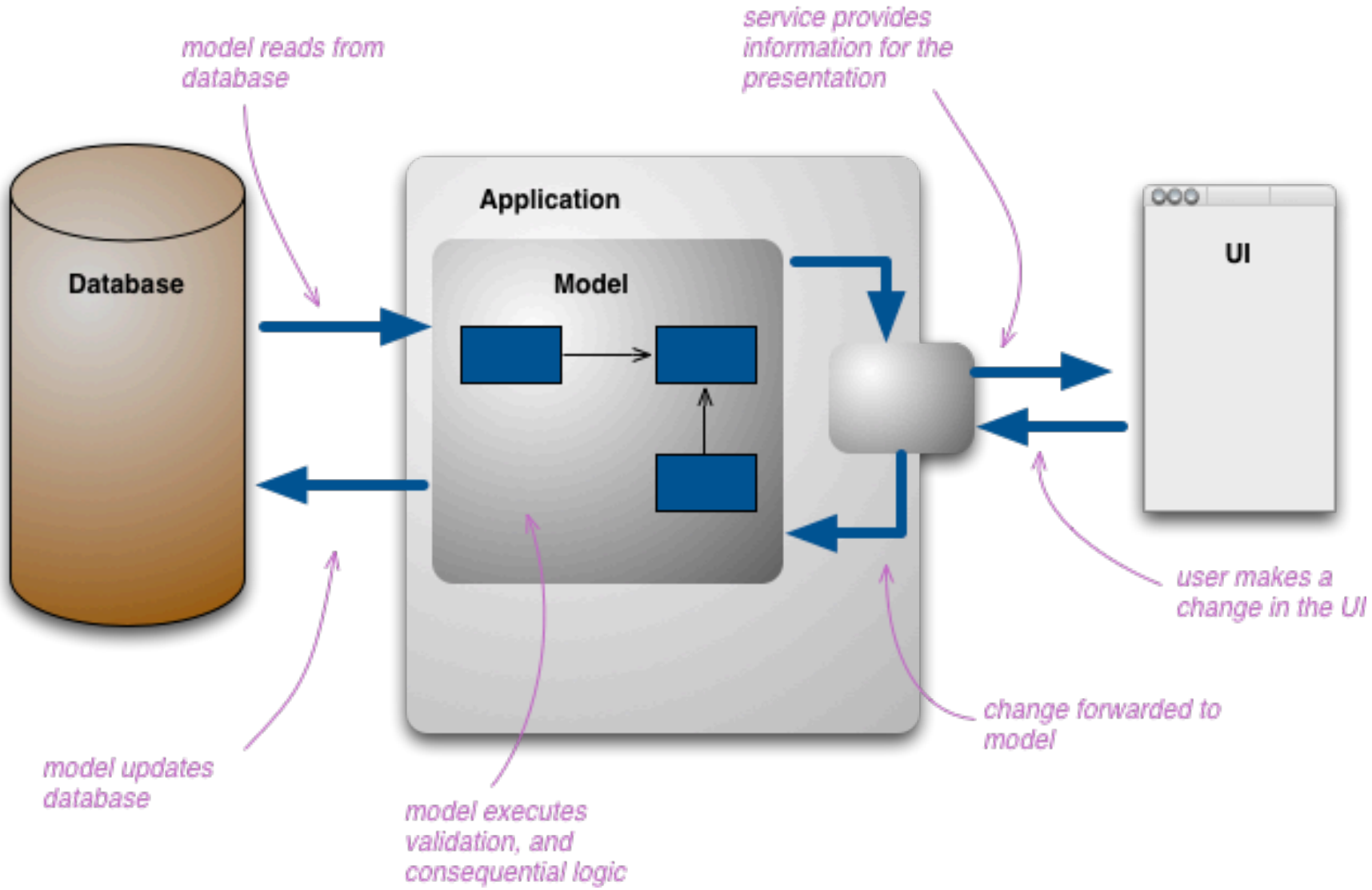
developed on a “dead end” branch

comparing architectures

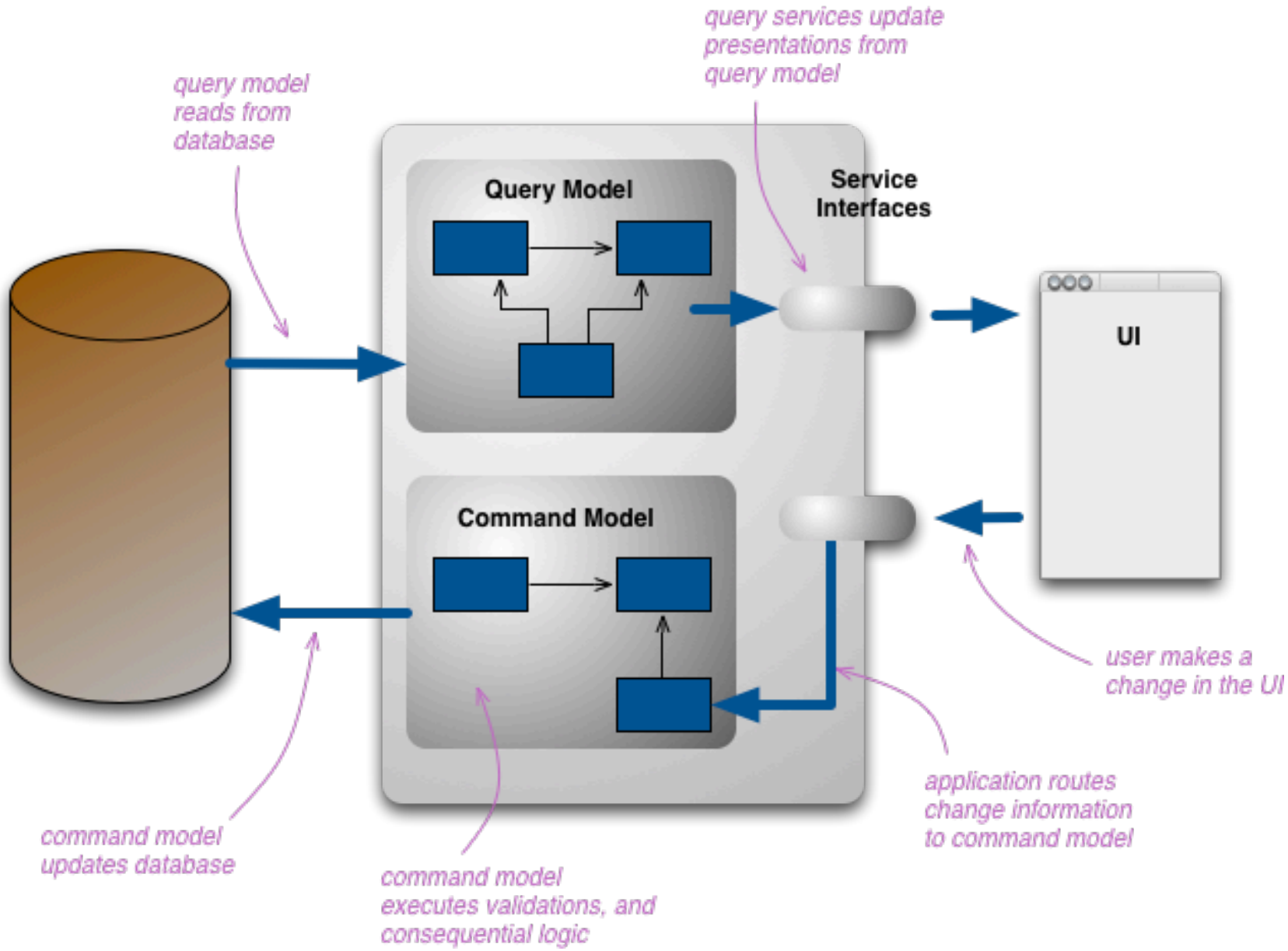





CQRS



traditional



CQRS Command Query Responsibility Separation

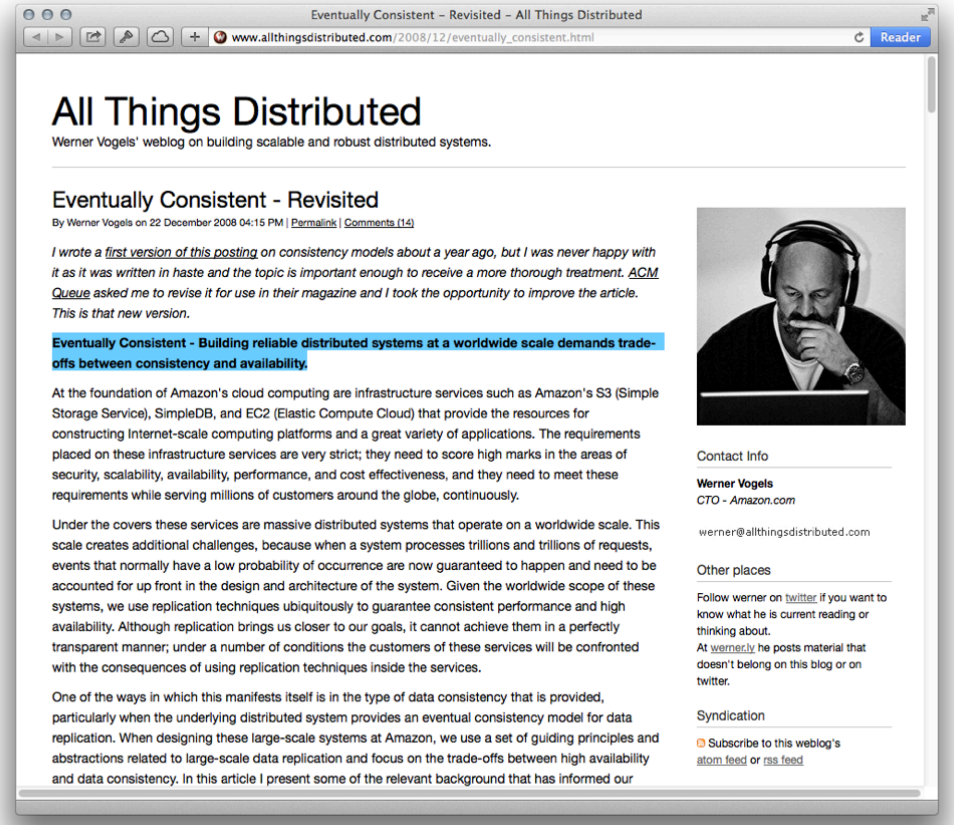
CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency

eventual consistency



"Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability."

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency

consistency or availability
(but never both)

complex or granular domains

domain logic classification

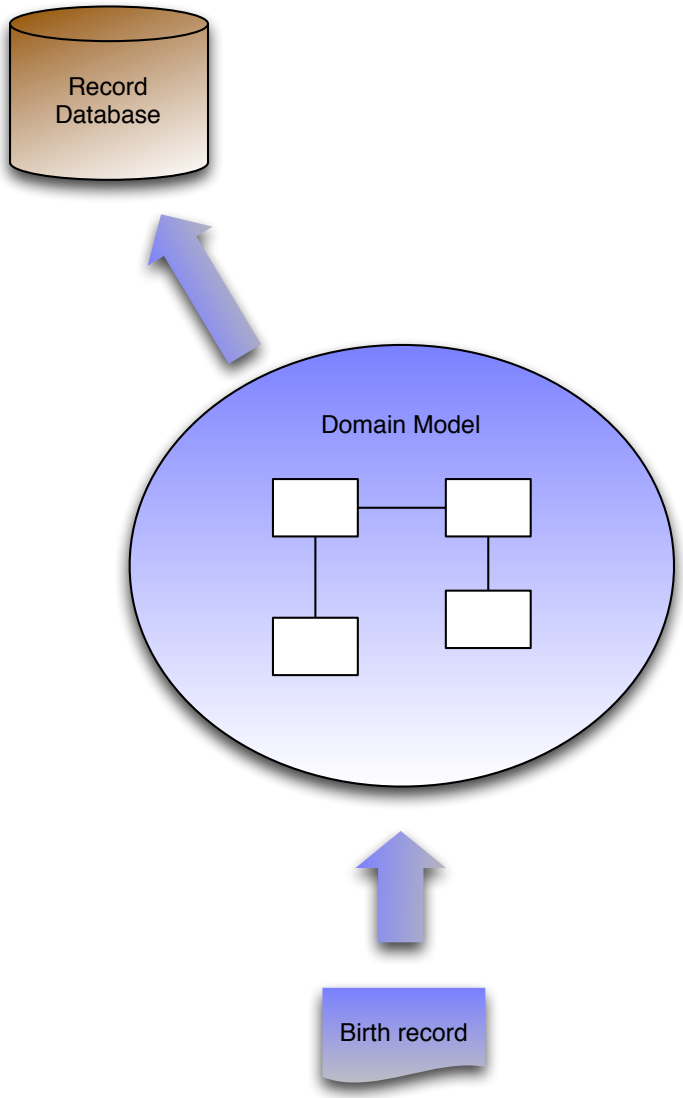
http://martinfowler.com/bliki/EagerReadDerivation.html

validations: sensical inputs, valid objects

consequences: initiating a world-changing action

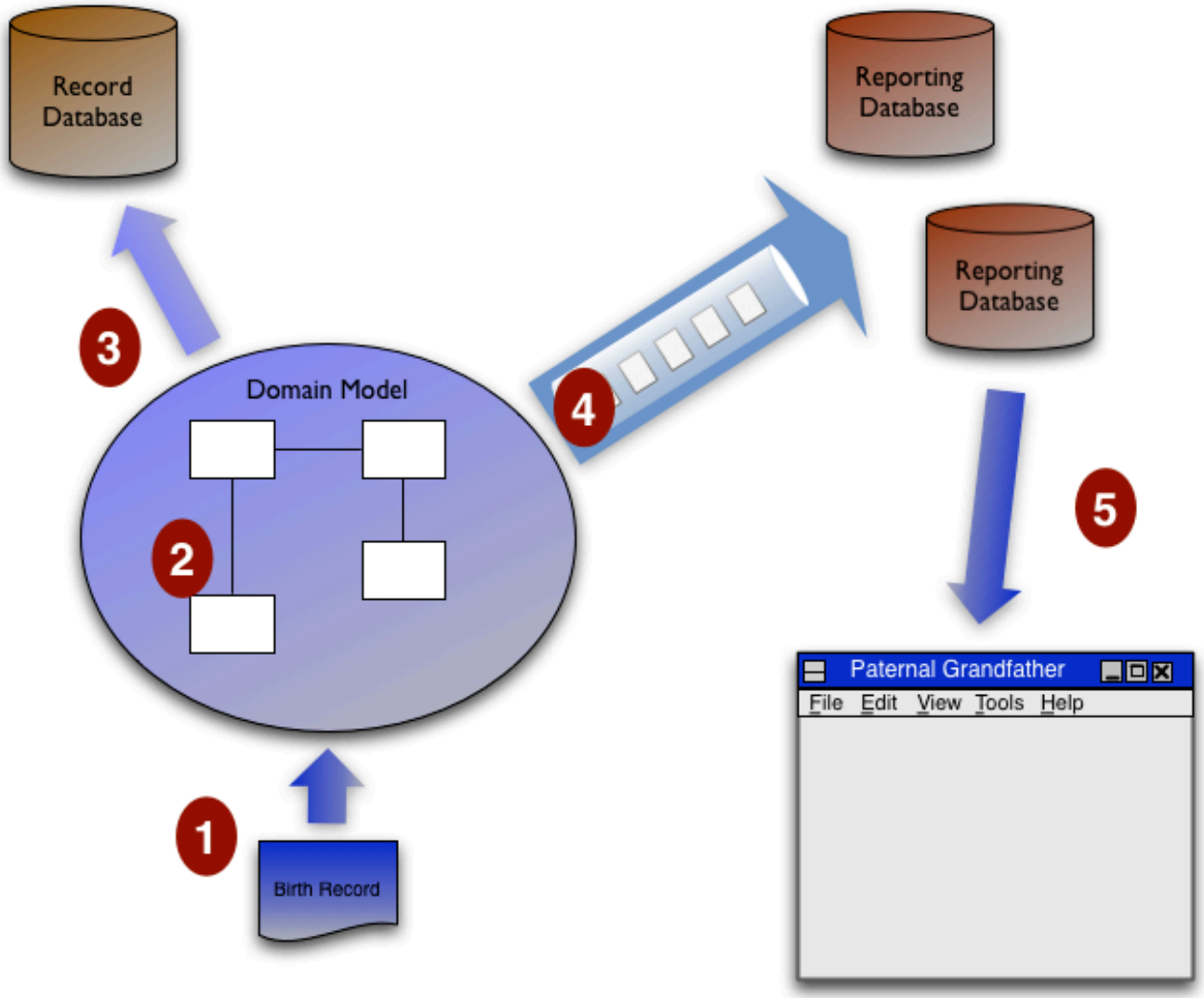
derivations: figuring out some information based on information we already have

typical case

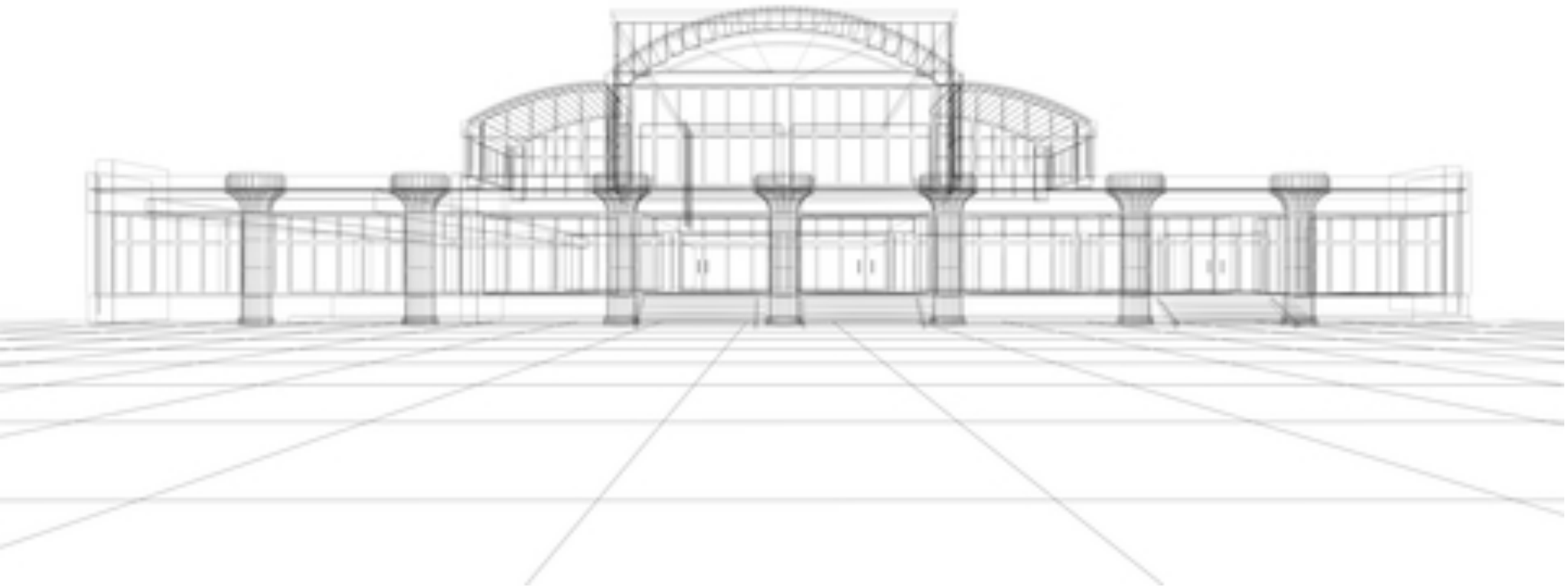


EagerReadDerivation

http://martinfowler.com/bliki/EagerReadDerivation.html



LMAX



LMAX

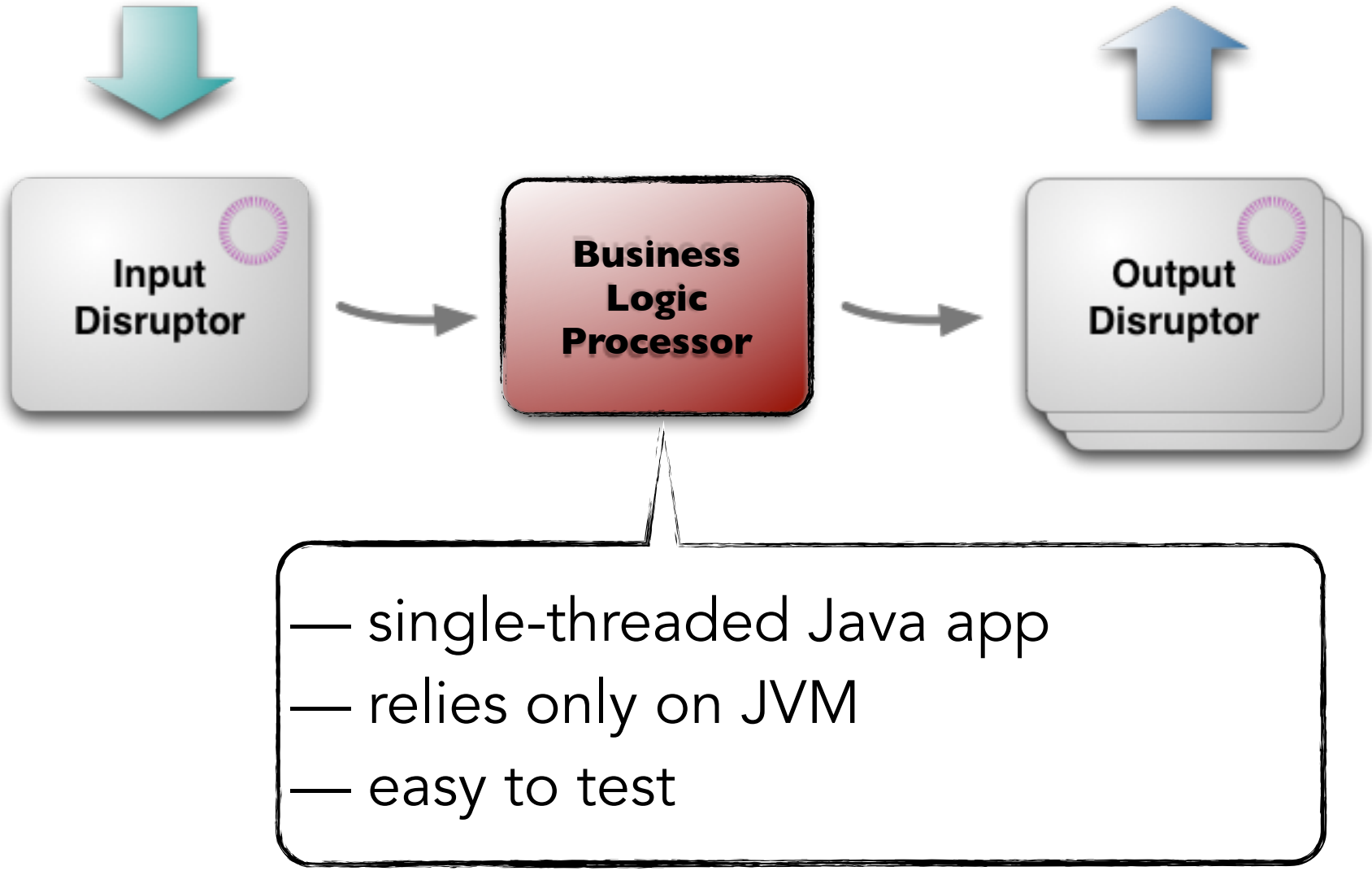
http://martinfowler.com/articles/lmax.html

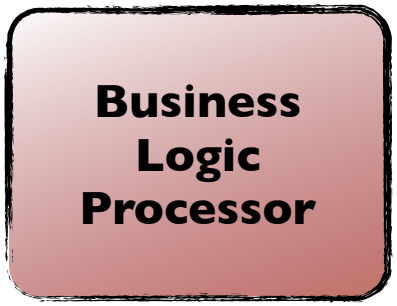
JVM-based retail financial trading platform

centers on Business Logic Processor handling 6,000,000 orders/sec on 1 thread

surrounded by Disruptors, network of lock-less queues

overall structure





business logic processor

in-memory

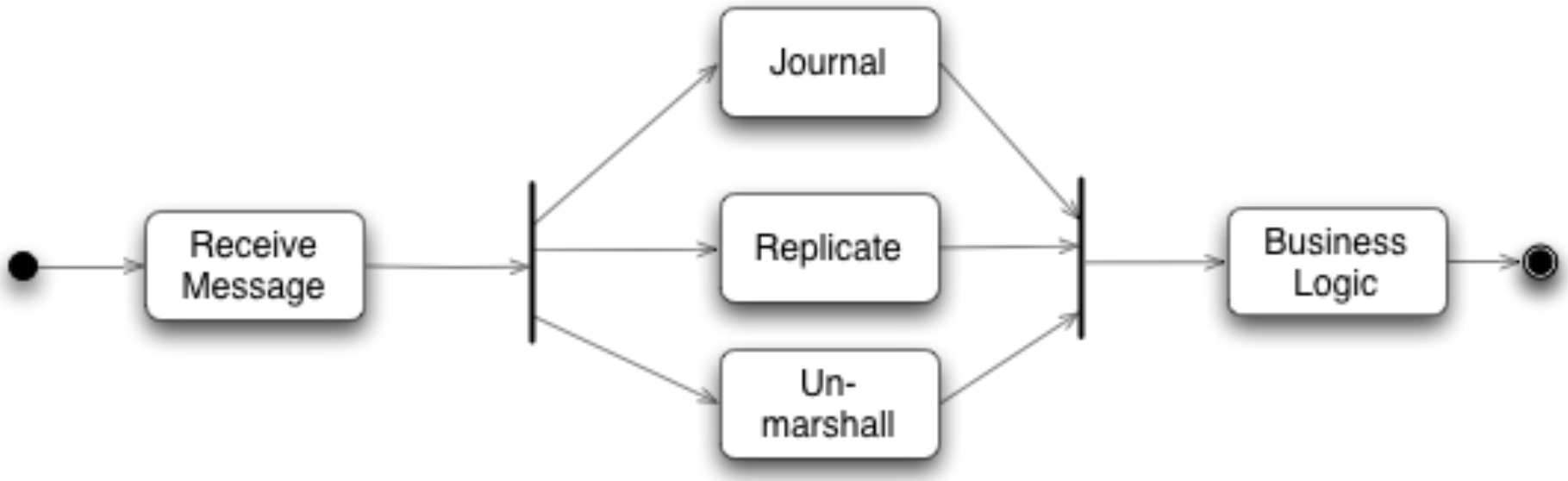
event sourcing via input disruptor

snapshots (full restart—JVM + snapshots—less than 1 min)

multiple instances running

each event processed by multiple processors but only one result used

input/output disruptors



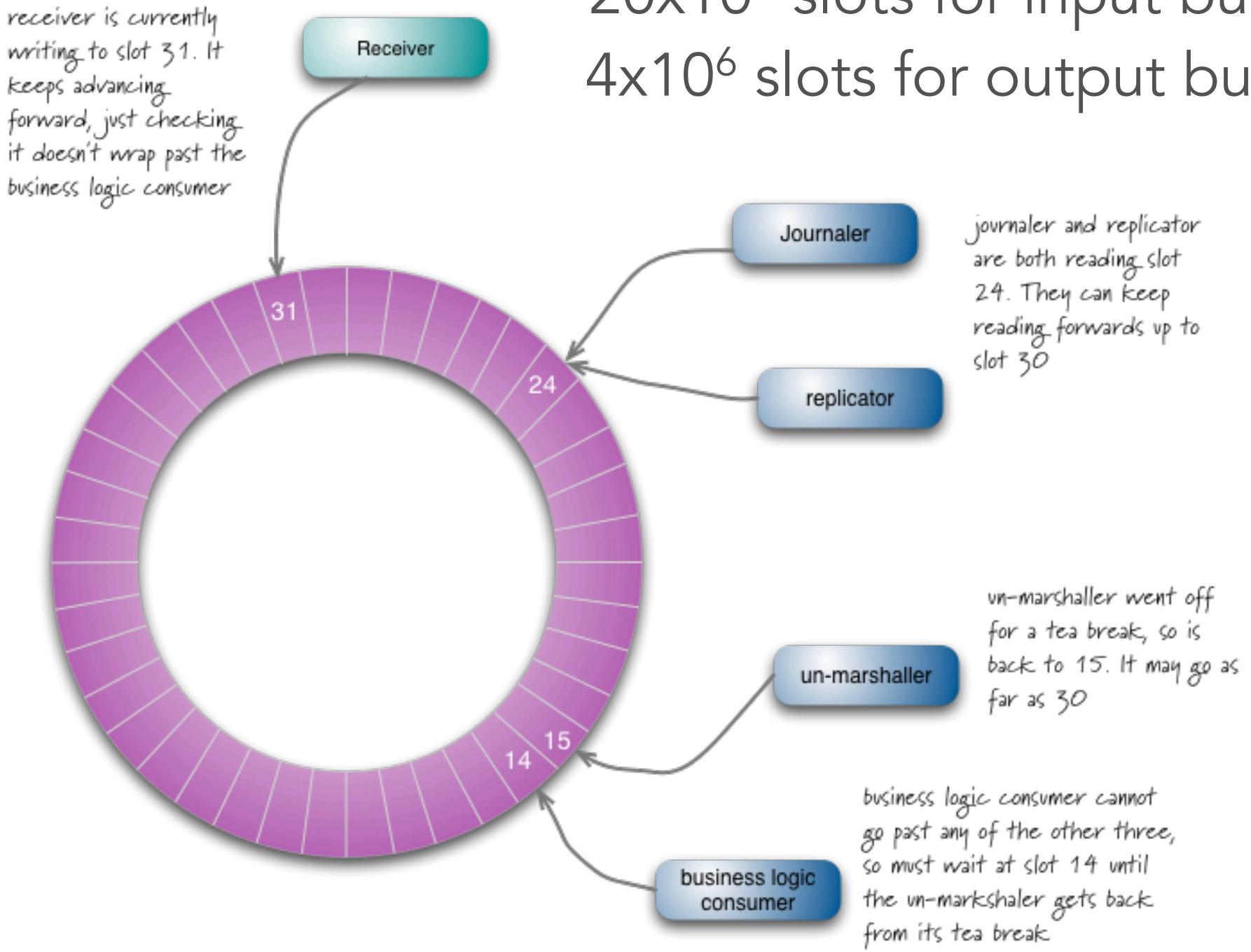
disruptors

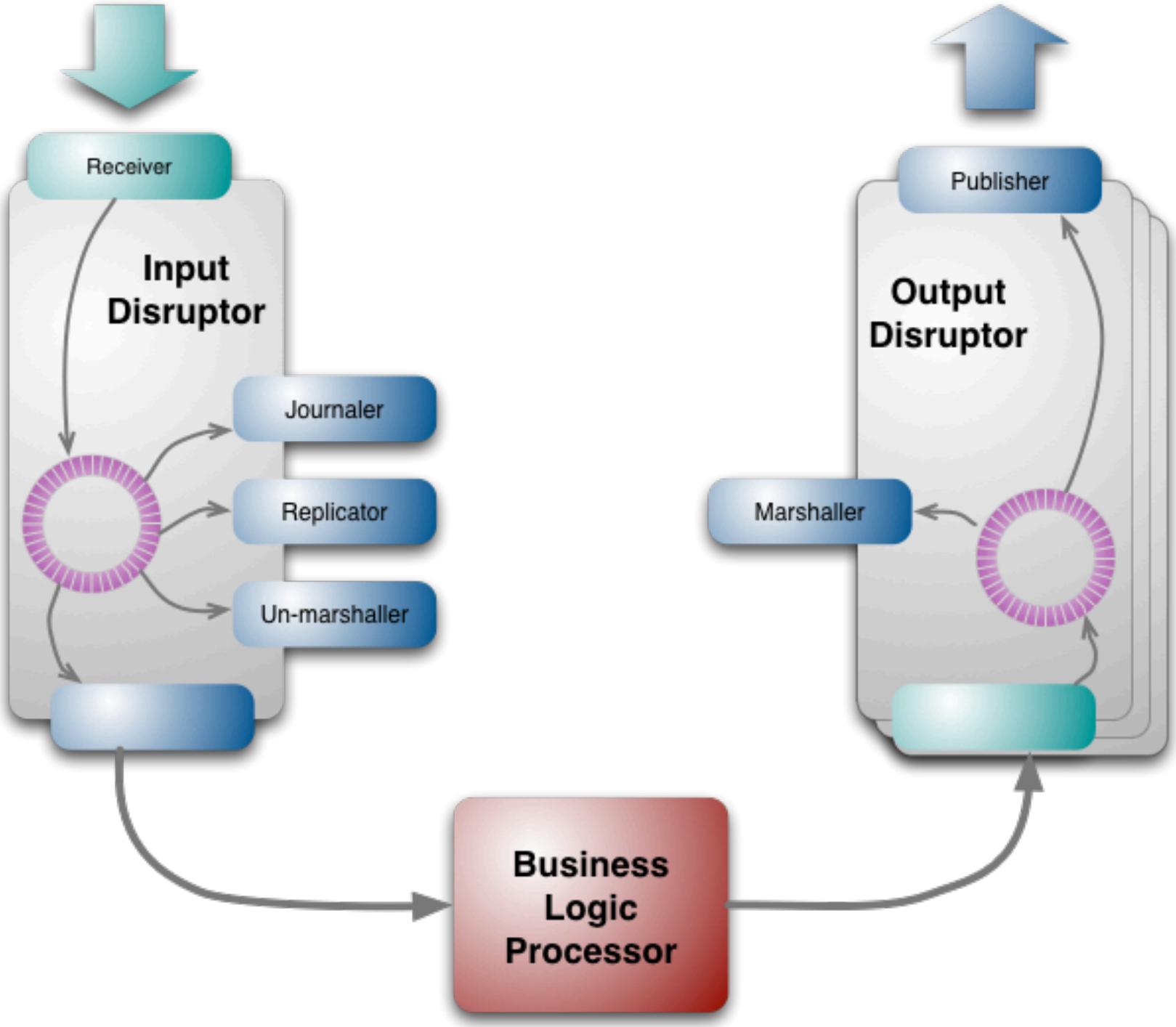
custom concurrency component

multi-cast graph of queues where producers enqueue objects and consumers dequeue in parallel

ring buffer with sequence counters

$20x10^6$ slots for input buffer
$4x10^6$ slots for output buffer




Receiver
Input
Disruptor
Journaler
Replicator
Un-marshaller
Business
Logic
Processor
Publisher
Output
Disruptor
Marshaller

“mechanical sympathy”

started with transactions

switched to Actor-based concurrency

hypothesized & measured results

CPU caching is key ➠ single writer principle

when would you use this architecture?

when would you avoid this architecture?

IT DEPENDS !

“One ring to rule them all…”

no architecture fits every circumstance

evaluate good, bad, and ugly

watch for primrose paths that turn ugly

embrace pragmatism

what's your day job?

building wicked cool architectures is an stimulating, rewarding intellectual challenge...

...building CRUD applications isn't

coolness sometimes == accidental complexity

architect for change vs YAGNI

?'S

Mark Richards
Independent Consultant
Hands-on Enterprise / Integration Architect
Published Author / Conference Speaker

http://www.wmrichards.com
http://www.linkedin.com/pub/mark-richards/0/121/5b9

Published Books:
Java Message Service, 2nd Edition
97 Things Every Software Architect Should Know
Java Transaction Design Strategies

Neal Ford
Director / Software Architect /
Meme Wrangler

ThoughtWorks®

2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA
T: +1 40 4242 9929 Twitter: @neal4d
E: nford@thoughtworks.com W: thoughtworks.com