goto; conference aarhus

IT'S ALL A NUMBERS GAME

THE DIRTY LITTLE SECRET OF SCALABLE SYSTEMS

Martin Thomson

@mjpt777

http://mechanical-sympathy.blogspot.com/

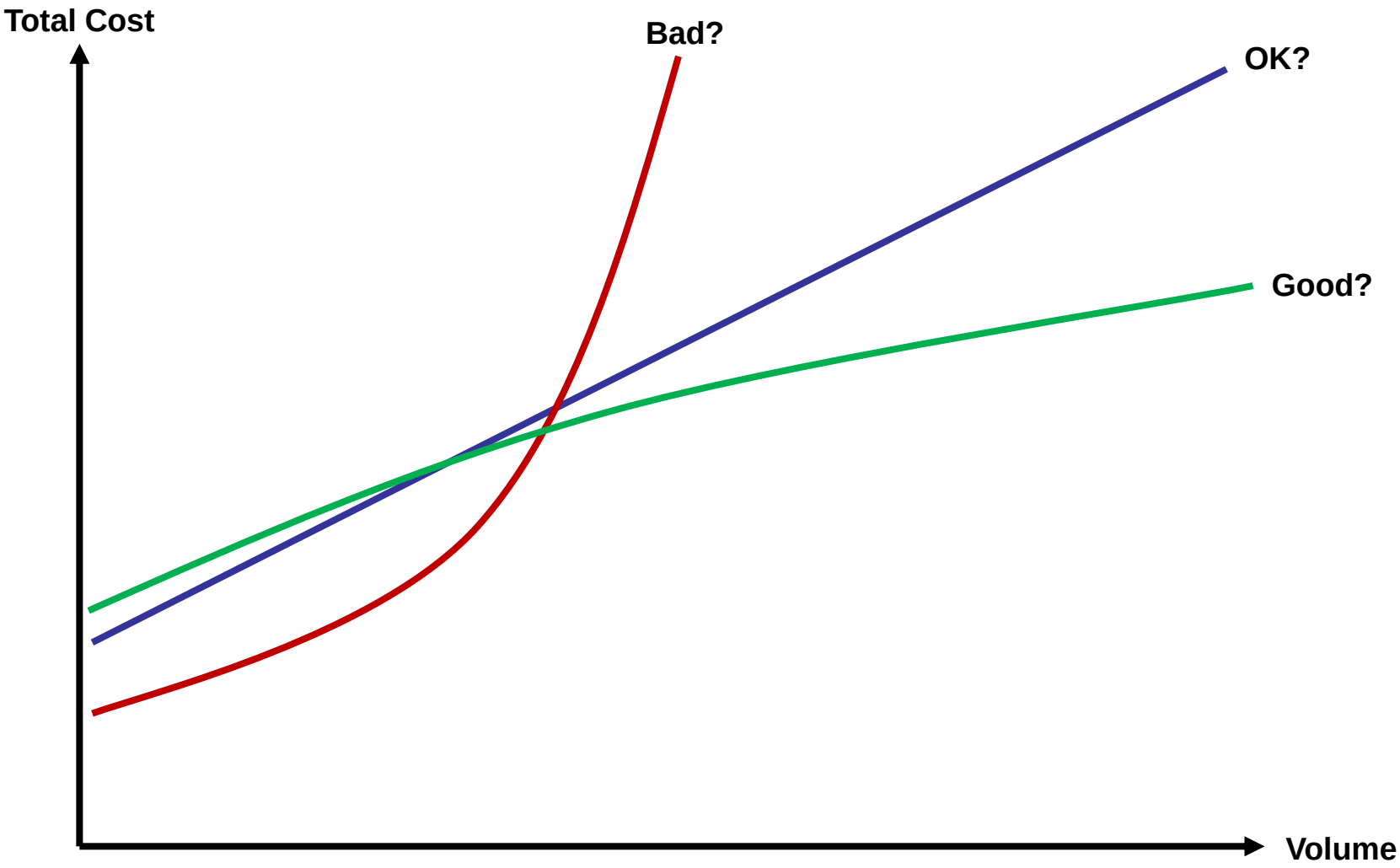
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com


SMOKING KILLS
RIZLA+
MILO
Cricket


DEBENHAMS
DEBENHAMS
IPANEMA
VANS
American Apparel
EVERYTHING
50%

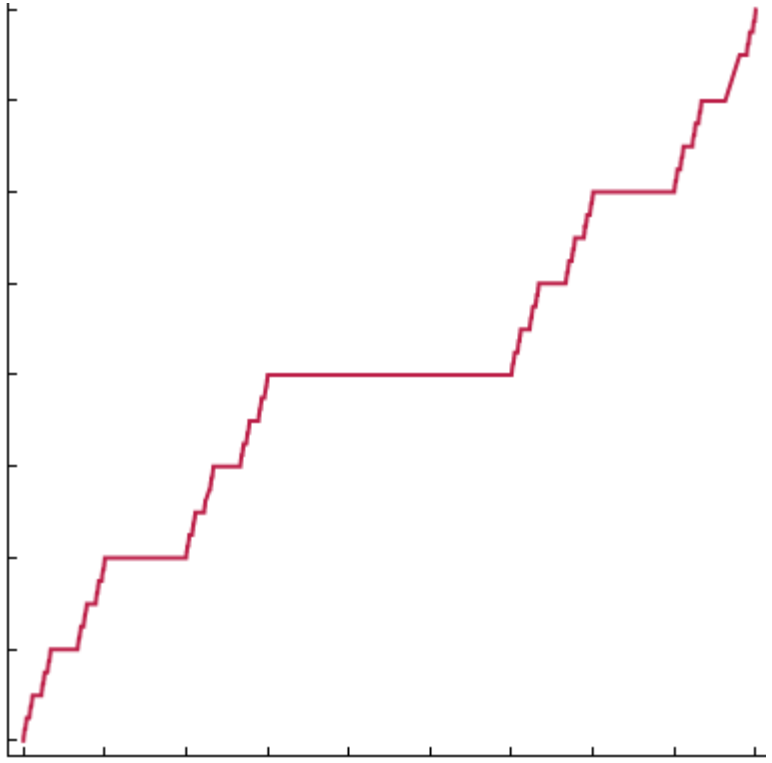
What does it mean to be Scalable?



It’s all about cost per Transaction

```
Transaction Cost = Total Costs / Transaction Volume
```

- **Fixed Costs**
  - > Upfront effort and infrastructure
  - > Need to amortize
  - > Capital vs. Operational spend
  - > How well do you know demand?
- **Variable Costs**
  - > Can they be on demand?
  - > Bulk discounts
  - > Guaranteeing available resources



How Many TPS Does An Additional Node Provide?

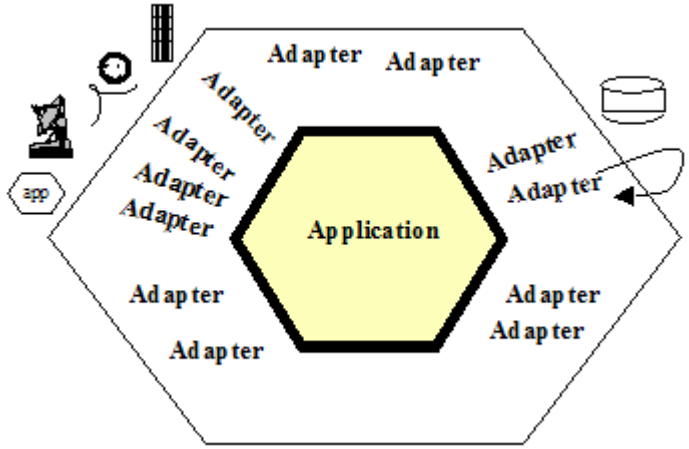
“Just throw hardware at the problem”

Guidelines for scalable systems

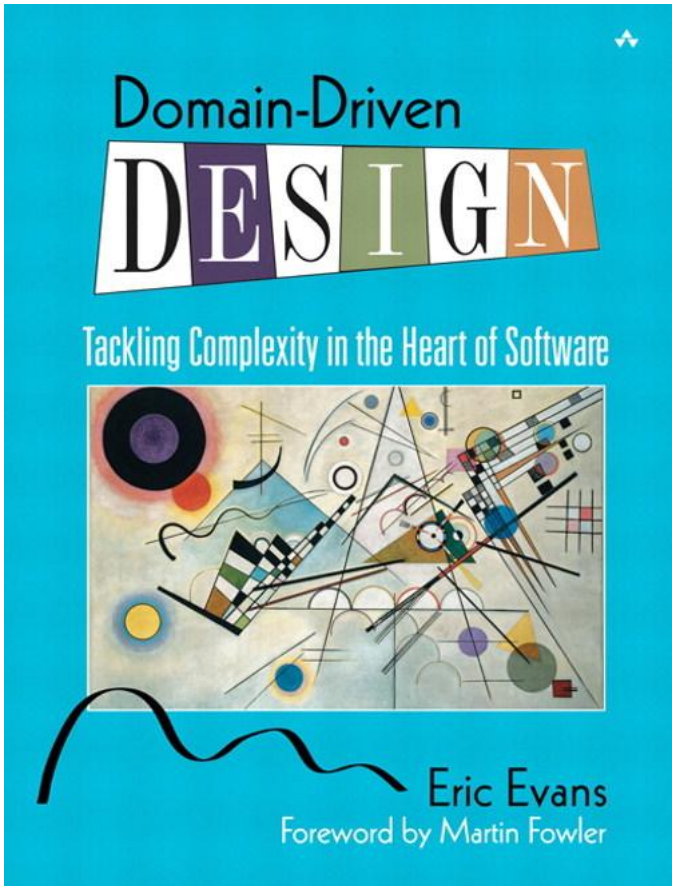
1. Domain Model at the Core
2. Performance Test & Profile
3. Understand Algorithm Behaviour
4. Eliminate Contention
5. Manage the Queues
6. Separate Reading and Writing
7. Know Your Platform/Infrastructure
8. Be Commercial

1. Domain Model at the Core

- **Pure model without any infrastructure**
- **Aggregates for clear entry points**
- **Minimal public interface**
- **Clean simple code!**
- **Layer around the core**



"*Hexagonal Architecture*"
- Alistair Cockburn



2. Performance Test & Profile

- Component Performance Tests
- System Performance Tests
- Production Monitoring
- Common performance test mistakes
- Theory of Constraints
- Drives the economics of a development

"Premature optimization is the root of all evil"
– Donald Knuth / Tony Hoare

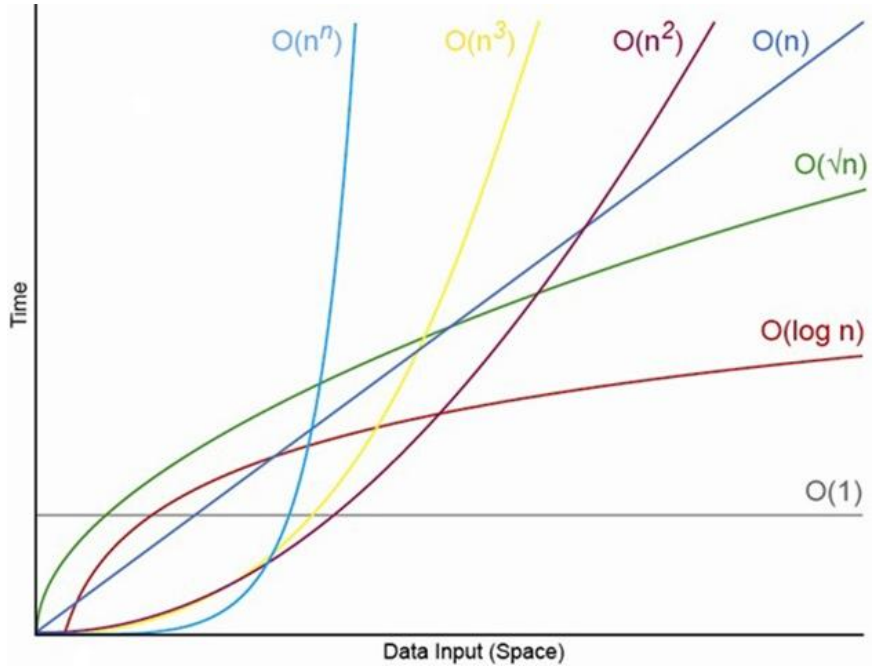
> **This is very different from knowing your capabilities, so test and profile early and often…**

3. Understand Algorithm Behaviour

- **Test cases with a set of size of 1**
  – *Really!*

- **Need to model realistic scenarios**
- **Model based on production**

- **Cache Oblivious Algorithms**
- **Unbounded queries are very bad**
  - > **Deal in manageable chunks**

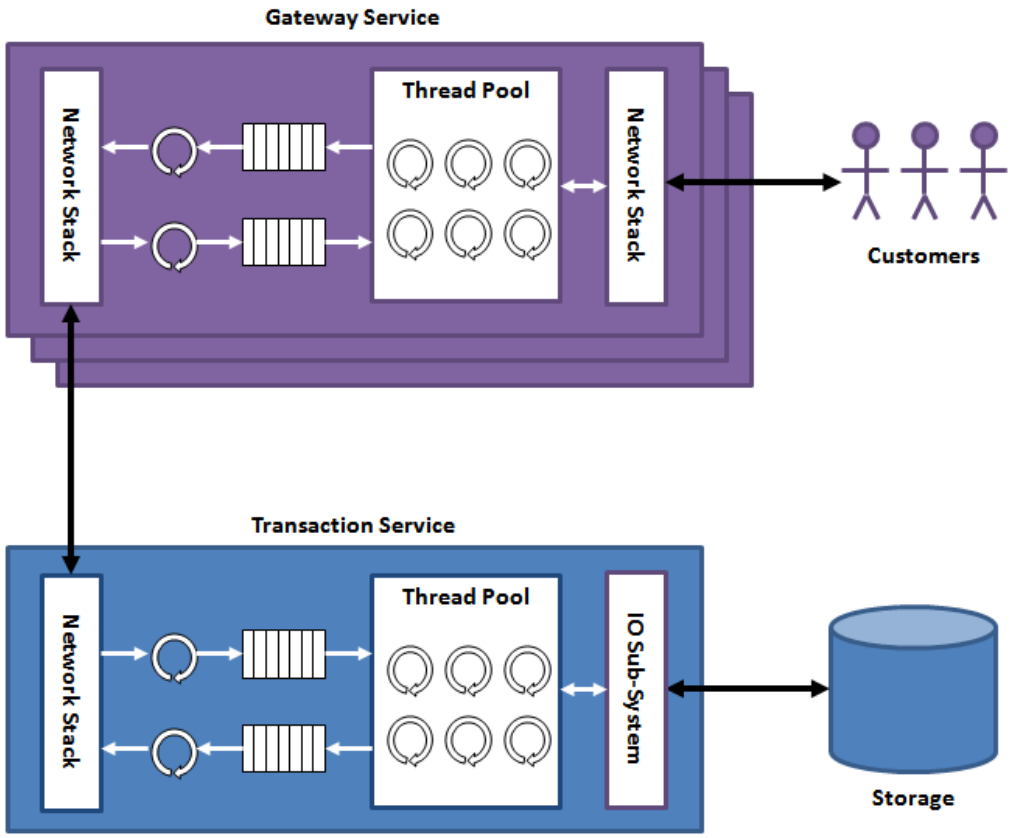


4. Eliminate Contention

- **Contention needs managed**
  - > **Management overhead often greater than actual work – e.g. locks**
- **Micro, Macro – all the same**
  - > **Lessons from the Disruptor**
  - > **Services and Databases**
  - > **“Load Balancers”**
- **Employ the “Single Writer Principle”**
- **Shared Nothing Architectures**
- **Design to allow sharding for writes**

5. Manage the Queues

- Little's Law
- Queues are everywhere!
  - > Make them explicit
  - > Keep them bounded
  - > Apply back pressure
- Queues manage contention but are also a source of contention
- Monitor queue lengths
- The *Curse* of Logging Libraries



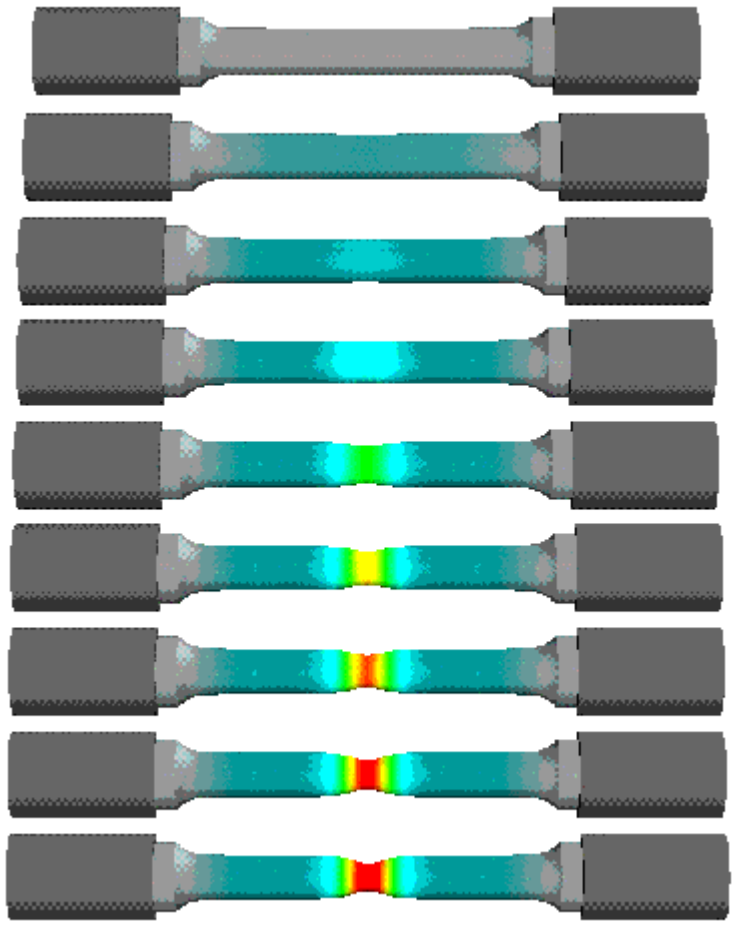
6. Separate Reading and Writing

- One of the best ways to relieve contention
  - Normally reads greatly outnumber writes
- Event Sourcing and CQRS
- Append Only Persistence
  - Even for traditional RDBMSs
- Caching
  - Reference Data
  - Fact based Data
  - Perfect != Right

7. Know Your Platform/Infrastructure

- **Mechanical Sympathy**
  - > **What are the platform capabilities?**
  - > **Operations Per Second**
  - > **Bandwidth**
  - > **Latency**

- **Load test until breaking point**
  - > **Do systems degrade gracefully?**
  - > **Do systems crash?**
  - > **Order of an algorithm?**
  - > **Failure and Replicas**



8. Be Commercial

- **Understand the Business**
  - > It is way more fun and rewarding
  - > Build a business using your great software
- **Never say, “No”**
  - > “Yes, and here are the consequences…”
- **Build relationships**
  - > Go for a coffee with others in the business
  - > Eat together
  - > Great Teams can be formed without formal structure
  - > Have fun!

Questions?

Blog: http://mechanical-sympathy.blogspot.com/

Twitter: @mjpt777