

# Advanced REST

Oxford University  
Software Engineering  
Programme  
May 2017



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Another way of looking at REST

## Taking HTTP seriously

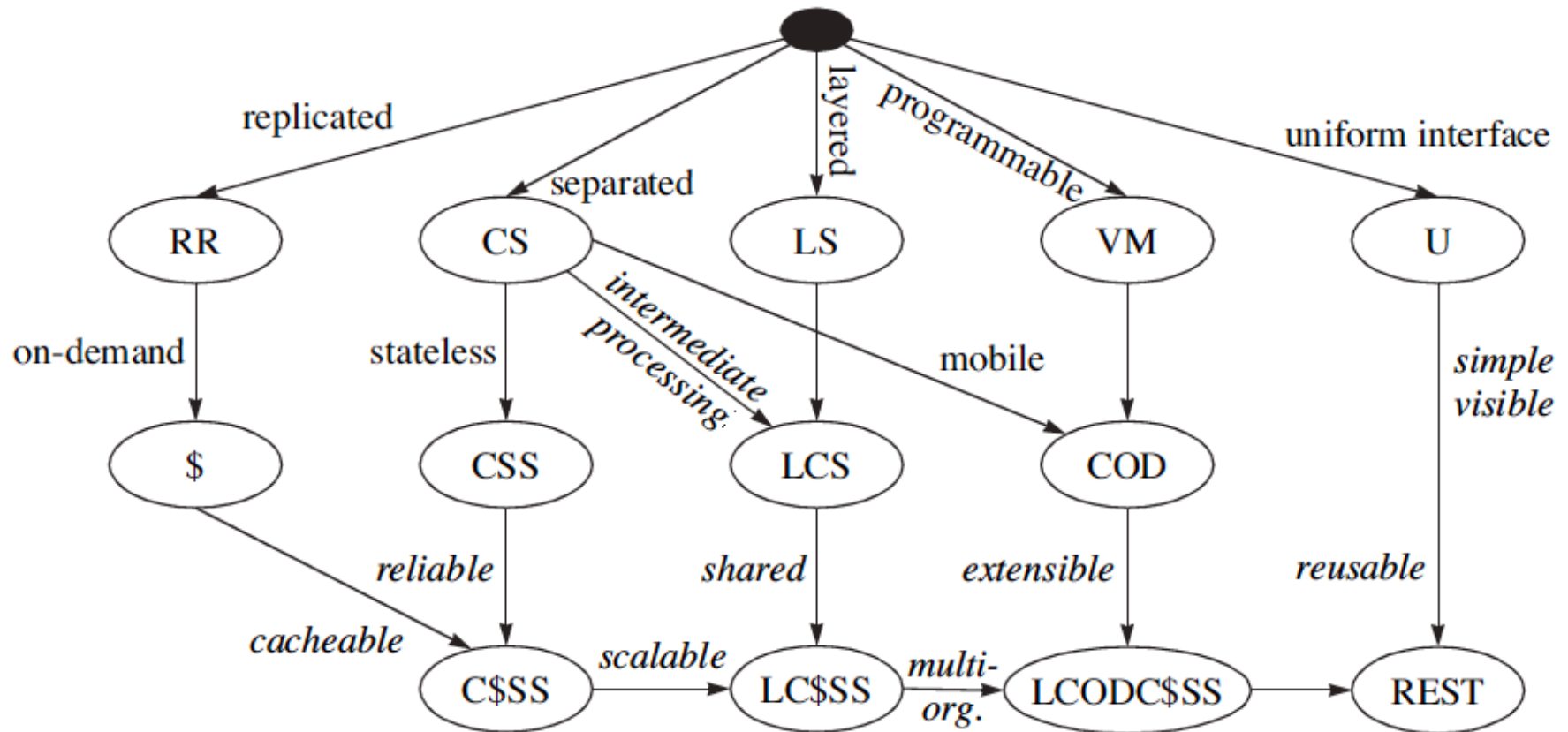


# REST

- **Roy Fielding**, a principal author of HTTP
- PhD thesis *Architectural Styles and the Design of Network-based*
- Subsequent article *Principled Design of the Modern Web Architecture* (ACM TOIT 2:2, 2002)
- Richardson & Ruby, *RESTful Web Services* architectural patterns of the web



# REST Derivation from Style Constraint



# Key concepts

- Client Server
- Cacheing
- Replicable
- Stateless
- Layered
- Uniform interface



# Cacheing

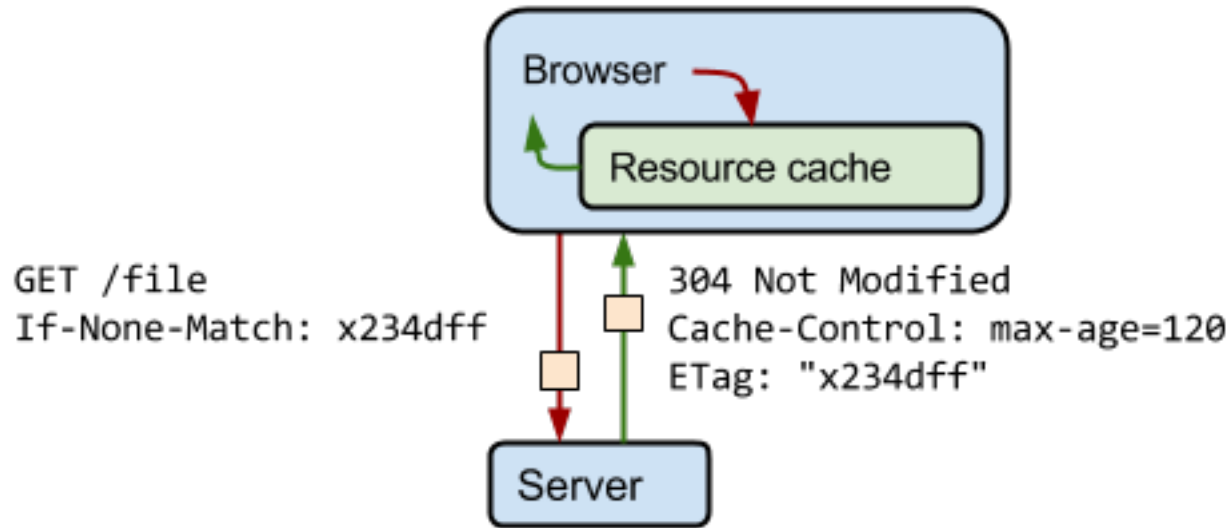
- Large scale network systems often rely on cacheing
  - Reduce traffic
  - Localised access
  - Reduced processing
    - **Akamai** and others make the web work effectively
- Caching relies on differentiating between cacheable and not cacheable traffic
  - Also understanding the lifetime and status of cacheable objects

# HTTP cacheing features

- Expires header
- Cache-control header
- If-modified-since / Not modified
- ETags (Entity Tags)
  - Uuids for content
  - Strong and Weak
- If None Match
  
- Unfortunately some websites are using Etags to track users instead of cookies!



# ETags



<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/images/http-cache-control.png>





© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Statelessness

- Of course there is state!
- The only question is where the state is kept.
- Traditional CS systems required the client and server to be kept in sync
- The web uses cookies



# scalability

/ˌskeɪləˈbɪlɪti/

noun

1. the ability of something, esp a computer system, to adapt to increased demands

Collins English Dictionary - Complete & Unabridged 2012 Digital Edition



# Speedup

- The **speedup** is defined as the performance of new / performance of old
  - e.g. move from 1 -> 2 servers
  - New system is 1.8 x faster than the old
    - In terms of transactions/sec (throughput)
  - Speedup = 1.8



# What inhibits speedup?

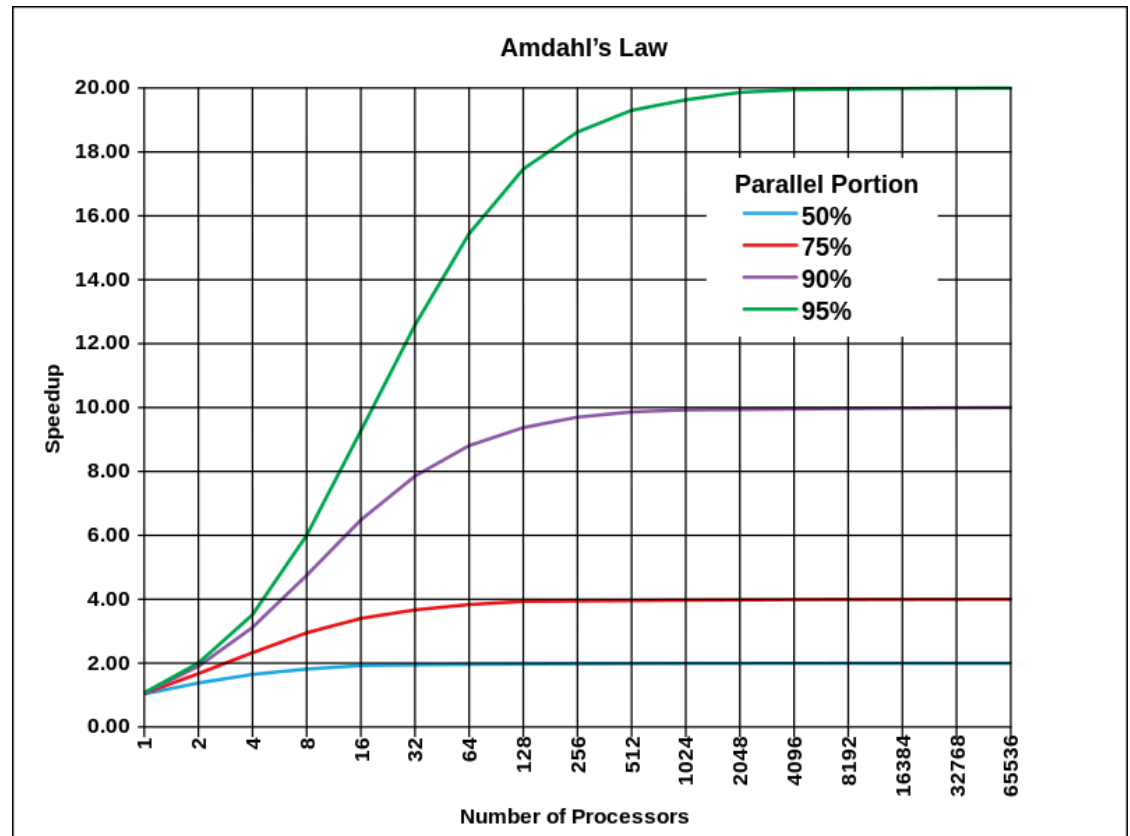
- In general you can split work into
  - Parallelizable and
  - Serial parts
- The serial parts stop you from scaling



# Amdahl's Law

Theoretical speedup given a fixed data size

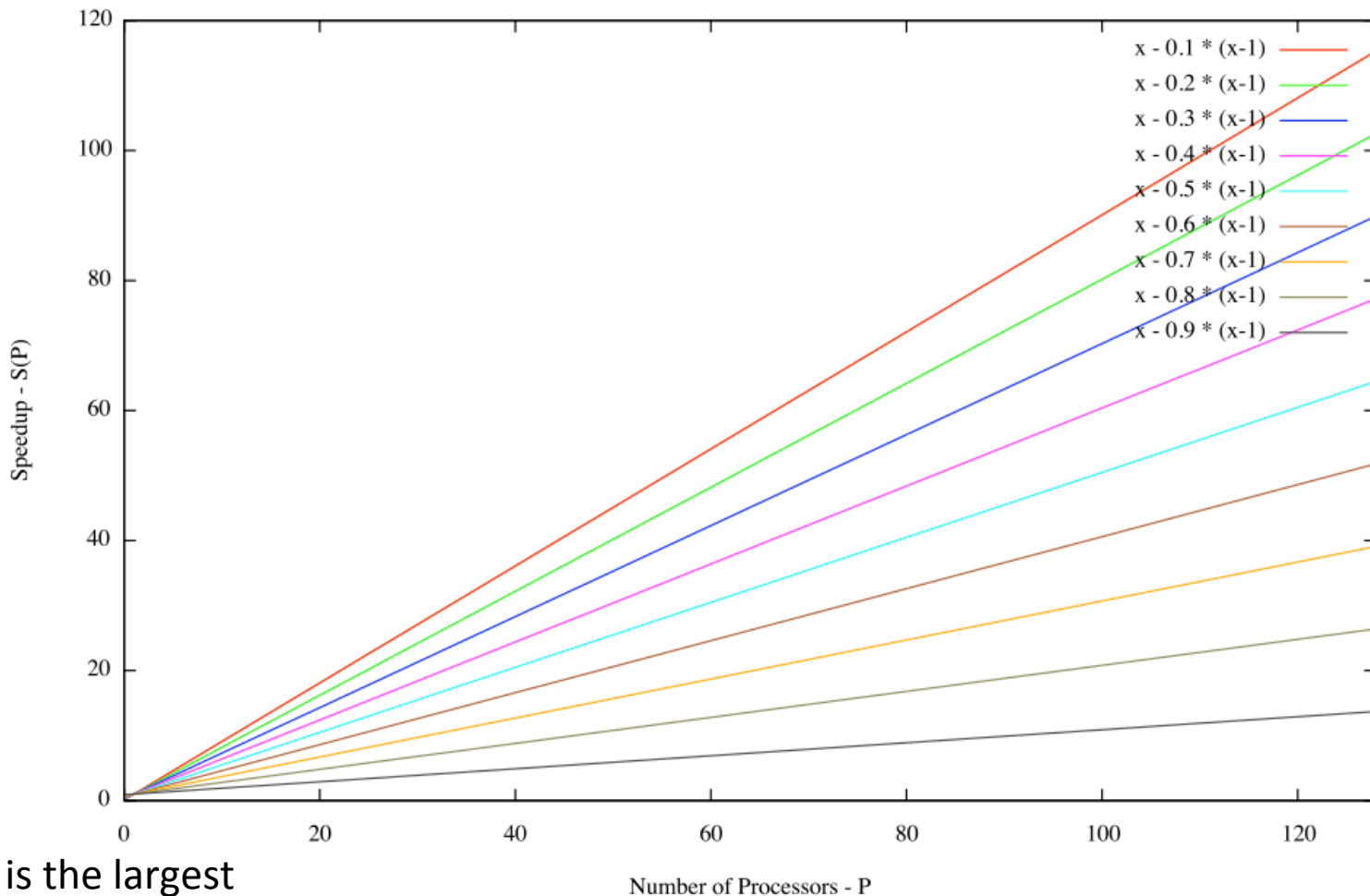
The speedup of a program using multiple processors in parallel computing is limited by the time needed for the serial fraction of the program, given a fixed size of data



# Gustafson's Law

What if the data increases too?

$$S(P) = P - \alpha \cdot (P - 1)$$



$\alpha$  is the largest

non-parallelizable fraction

# A driving metaphor

- **Amdahl's Law**

- You are travelling to London (60 miles)
- 30 miles in you have spent one hour
- You can never average  $> 60$  mph

- **Gustafson's Law**

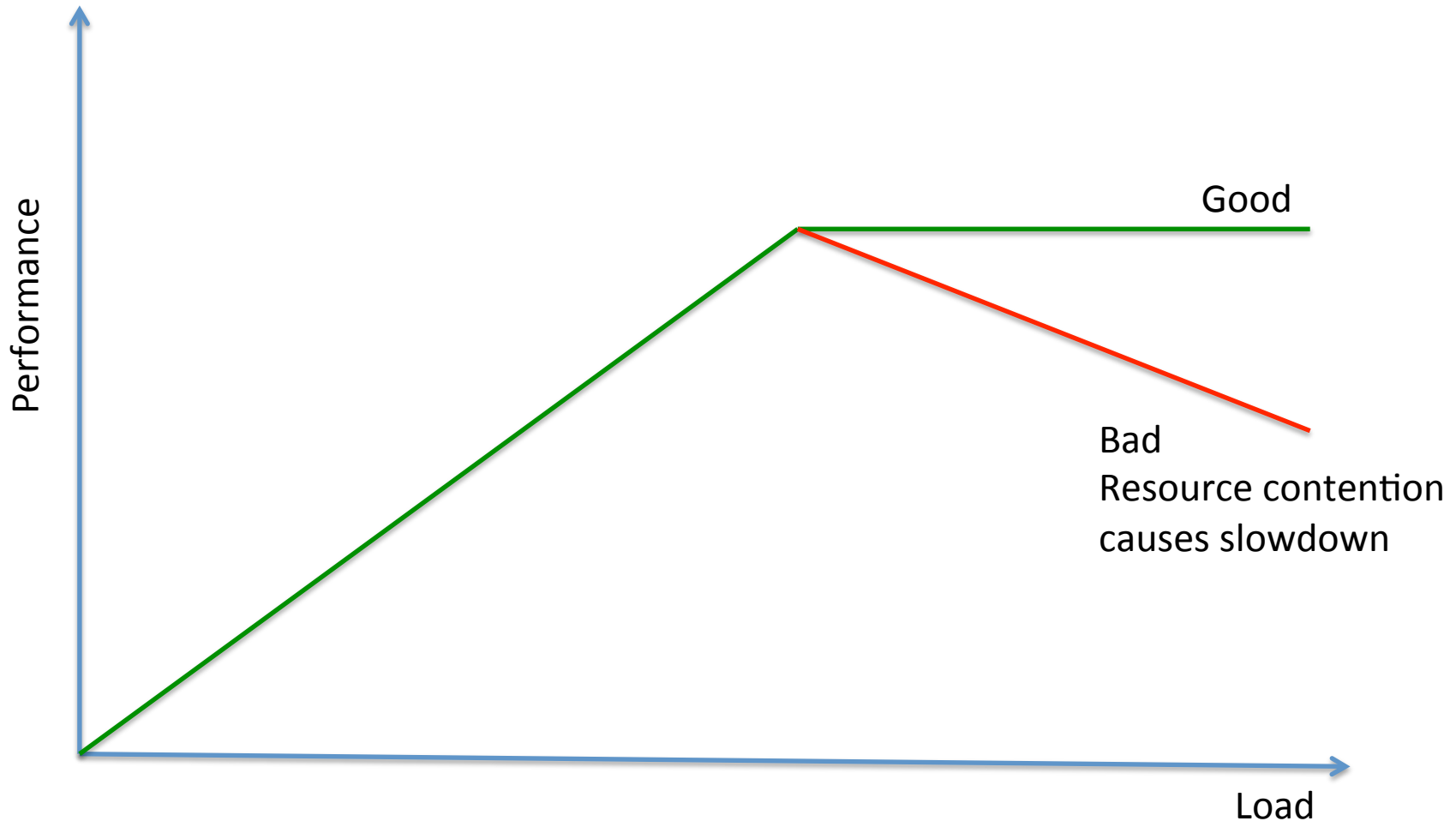
- You are travelling across the US
- You've spent an hour at 30 mph
- You can achieve any average speed given enough time and distance





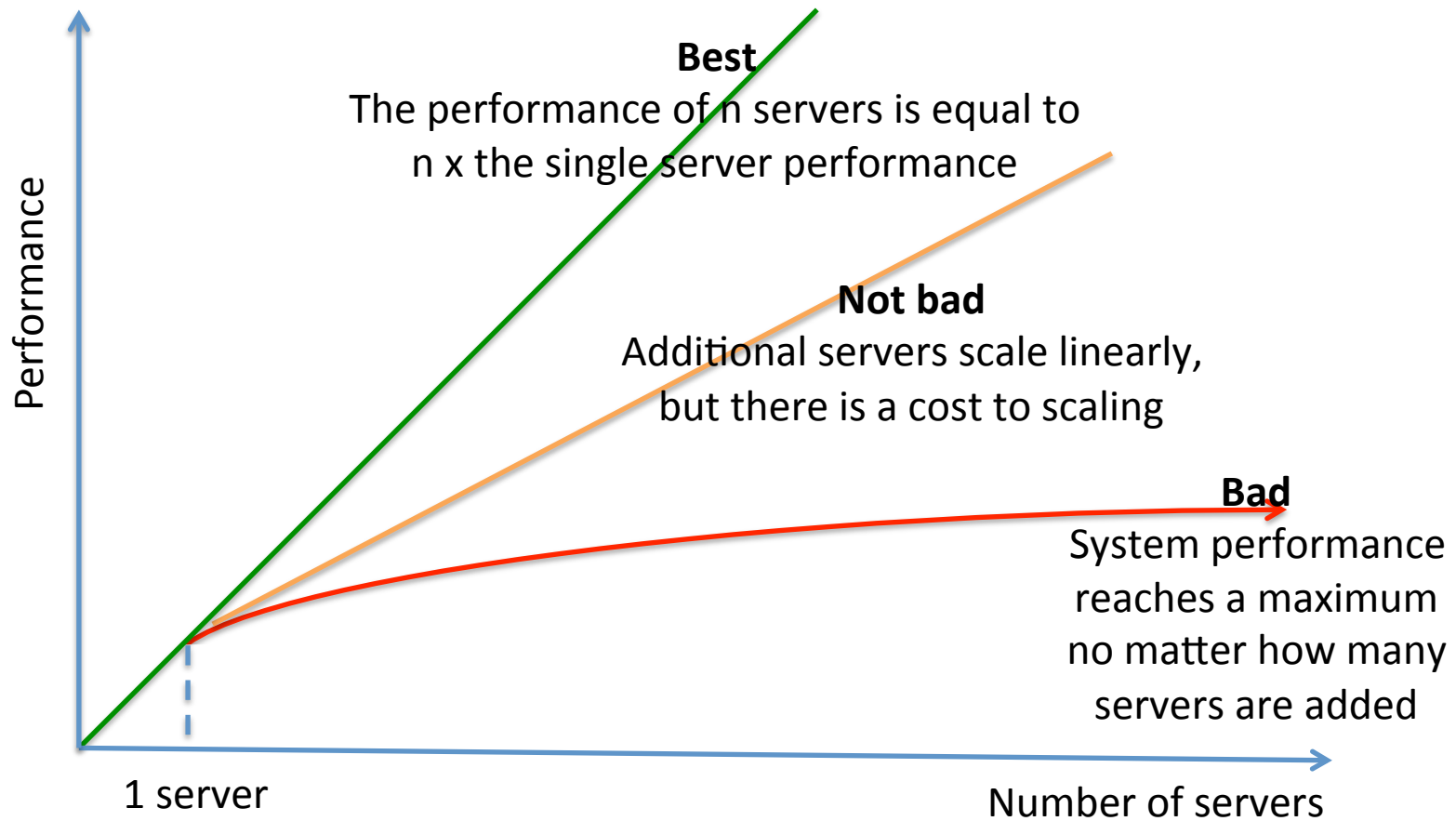
# Performance

Single system under increasing load



# Performance

## Scaling servers when fully loaded



# Karp-Flatt Metric

$e$  is the Karp-Flatt Metric

$\psi$  is the speedup

$p$  is the number of processors

$$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$

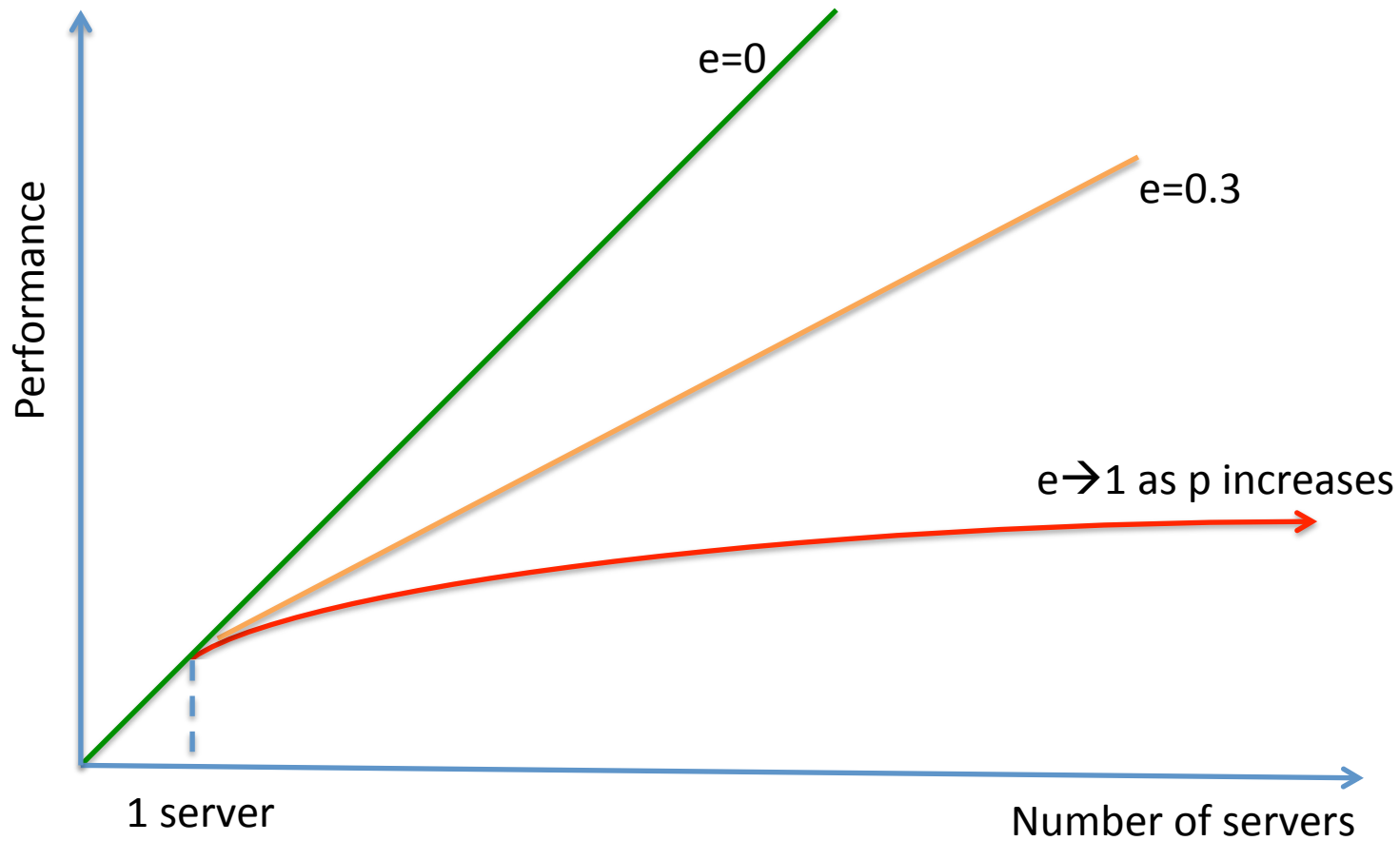
$e = 0$  is the best

$e = 1$  indicates no speedup

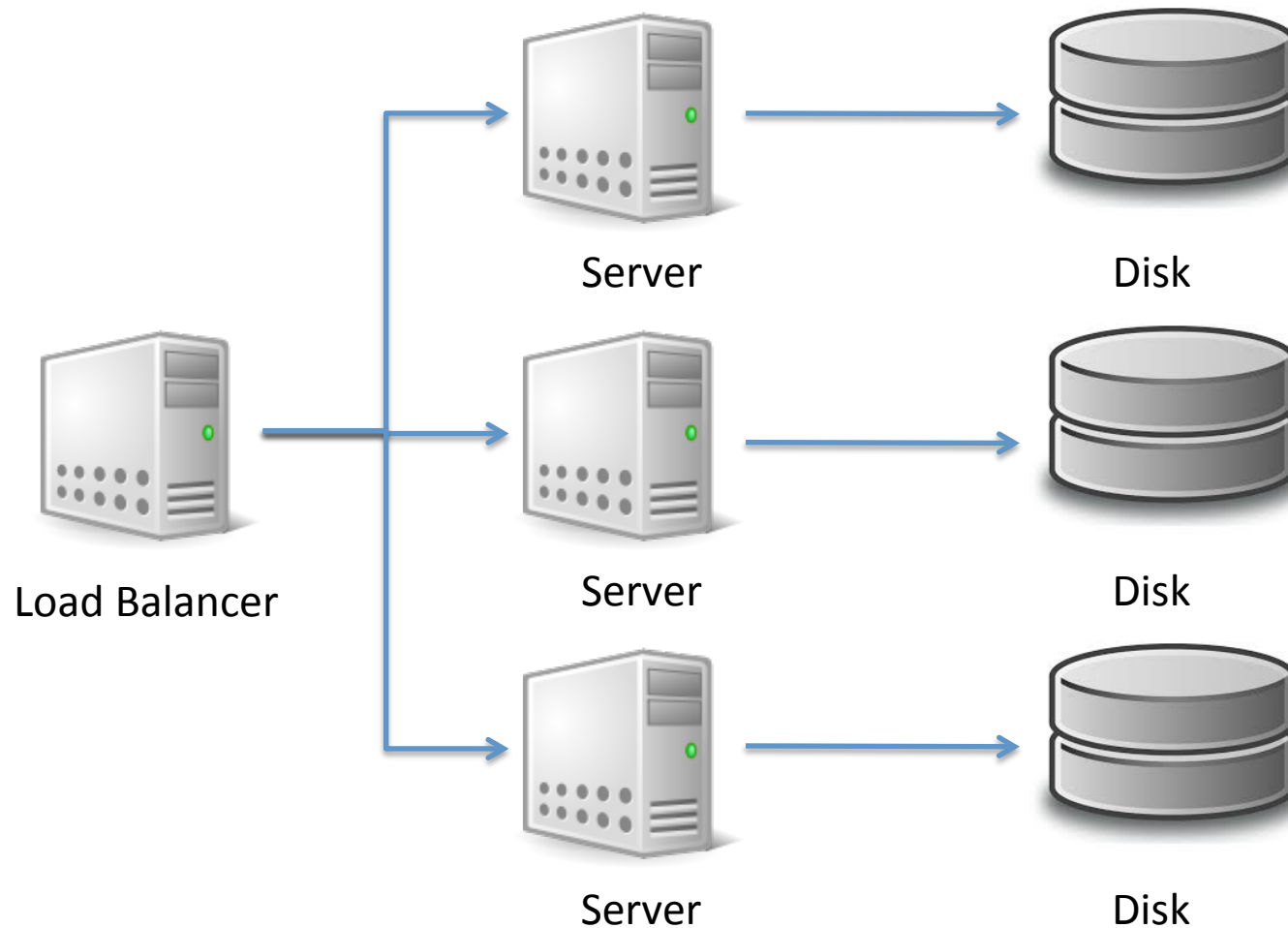
$e > 1$  indicates adding processors

slows down the system!!!

# Karp-Flatt metric



# Shared Nothing Architecture

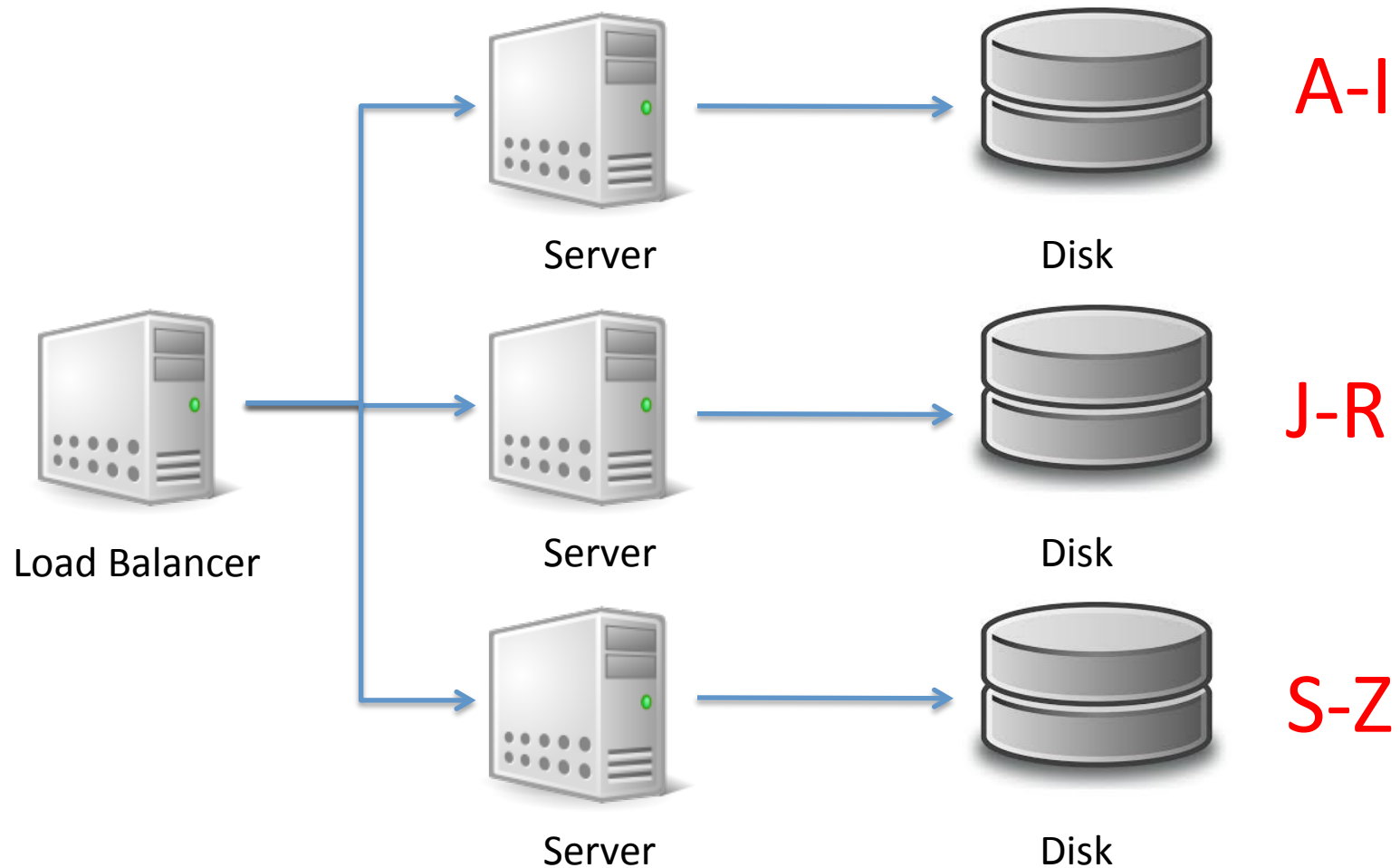


# Shared Nothing Architecture

- Implies there is no serial part to the computation
- Karp-Flatt Metric of 0
  - Assuming 100% efficient load balancing
- In practice, this is difficult!



# Partitioning / Sharding



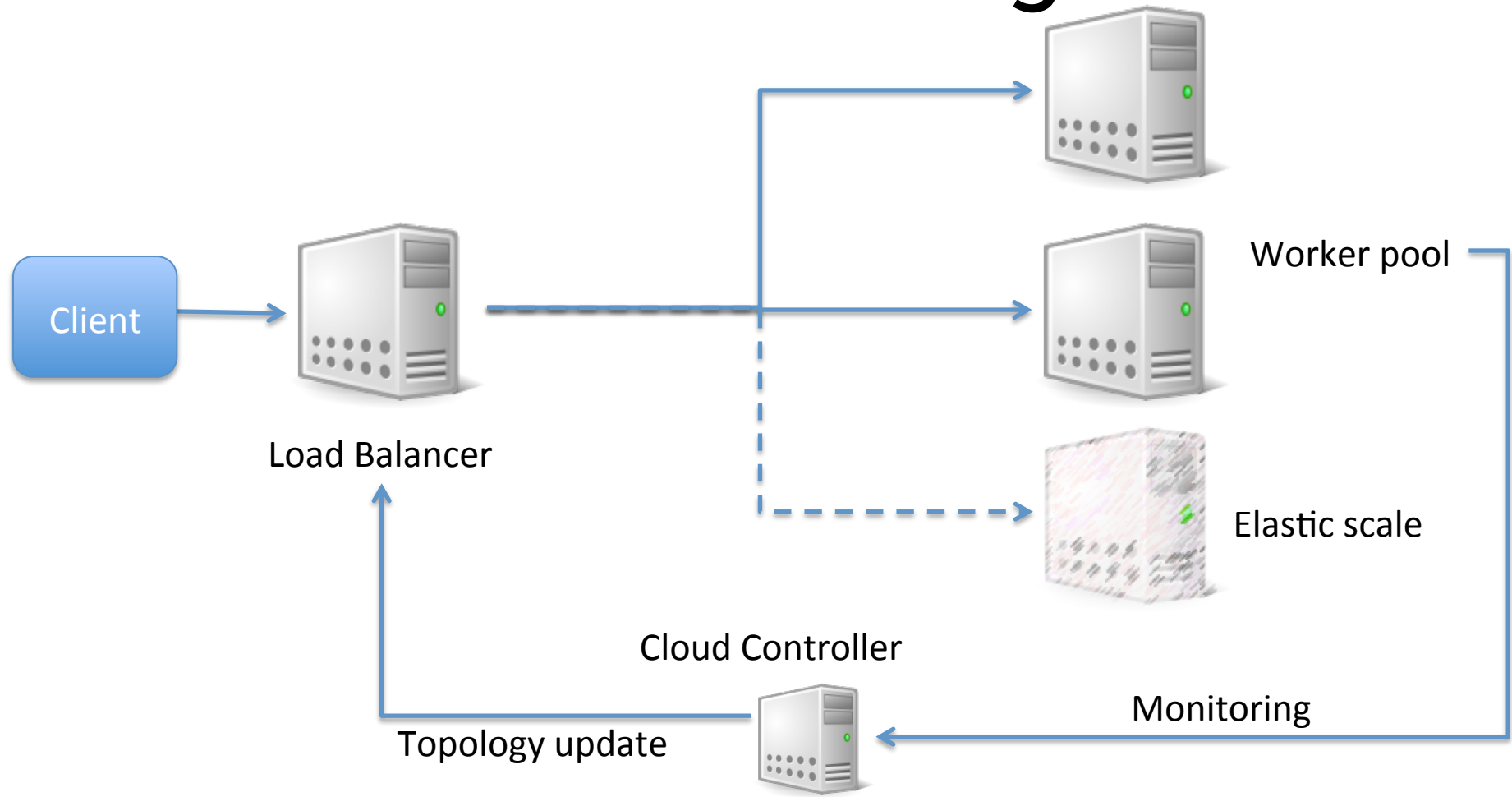
# Problems with Sharding

- Imbalance
  - Fewer S-Z's than A-I's
- Failover
- Adding new servers requires a re-balance
  - Is this automatic or manual?!





# Load Balancer-based elastic scaling



# Statelessness is hard

- There is a lot of intermediate calculation in most web systems that needs to be stored between transactions.
- The exemplar is the shopping cart

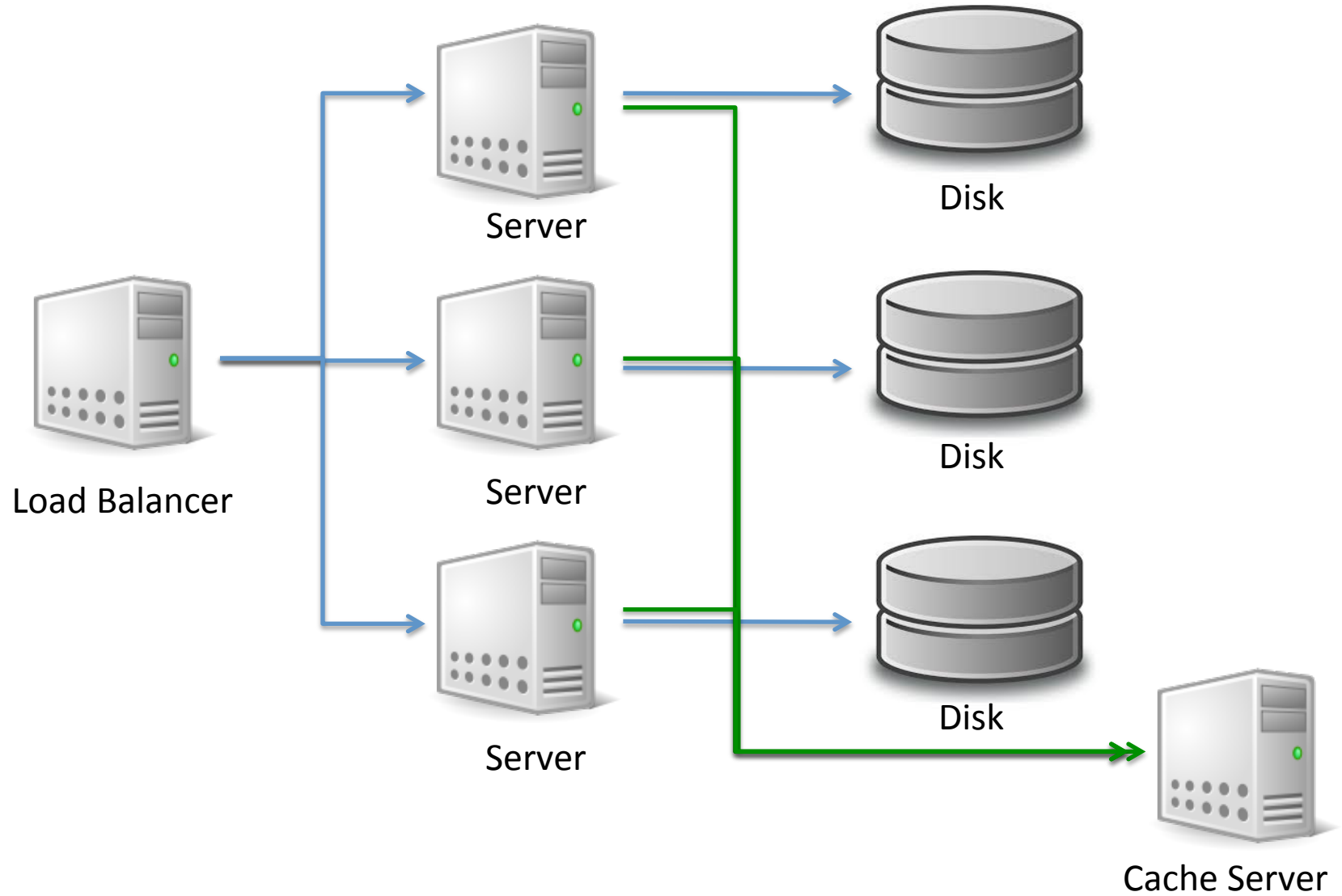


# Intermediate state storage

- In-memory
  - Breaks statelessness, limits scalability
- In the client
  - OK for some APIs but can be slow and hard to program
- In the database
  - Slow and expensive
- Cache servers:
  - The client keeps a cookie, which is the key to the datastore
  - The usual practise
  - Redis, memcached, Hazelcast, Infinispan



# Cache



# Layered systems

- Reverse proxies
- Composable
  - E.g. my business process is a service that exposes a REST interface and coordinates other services
- Compare with web scraping



# Principles of REST Architecture

- REST isn't protocol specific, but in practice means the RESTful usage of HTTP
  - CoAP is an example of a non-HTTP restful interface
- HTTP is very rich:
  - Content negotiation
  - Distributed caching.
- HTTP verbs nicely map to CRUD operations of data
- RESTful web services
  - Try to use all of HTTP as an application protocol



# Resources and Uniform Interface

- Addressable Resources. Every “object” on your network should have a unique ID.
- An important aspect is that each “object” or resource has its own specific URI where it can be addressed
- The URI should have a lifetime equivalent to the resource it represents
  - (I’ve had the same bank account for 20+ years)



# Representation

- State of resource captured and transferred between components
- Might be current or desired future state
- Represented as data plus metadata (name-value pairs)
- Metadata includes control data, media type
- The **Content-Type** of the resource should be useful and meaningful (self-description)
- One resource might have several representations
- Selected via separate URIs, or via content negotiation





# Uniform Interface

- A Uniform, Constrained Interface. When applying REST over HTTP, stick to the methods provided by the protocol
  - GET, POST, PUT, and DELETE.
- These should be used properly
  - GET should have no side effects or change on state
  - PUT should update the resource “in-place”



# REST Standards

- **HTTP 1.1**
- **URI**
- **URI Template**
- **WebSockets**
- **XML, JSON, etc**
- **Atom/AtomPub**
- **OData**
- **OpenId**
- **OAuth 1 / 2**
- **SAML/SAML2**
- **JSON Web Tokens**
- **WADL**
- **Swagger**
- **Json Home**
- **Json Web Encryption**
- **Json Web Signature**
- **Json Patch**
- **SPDY**
- **HTTPbis**
- **HTTP Link Header**
- **Microformats**
- **RDDL**
- ...



# HTCPCP 1.0

Network Working Group  
Request for Comments: 2324  
Category: Informational

L. Masinter  
1 April 1998

Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1998). All Rights Reserved.

Abstract

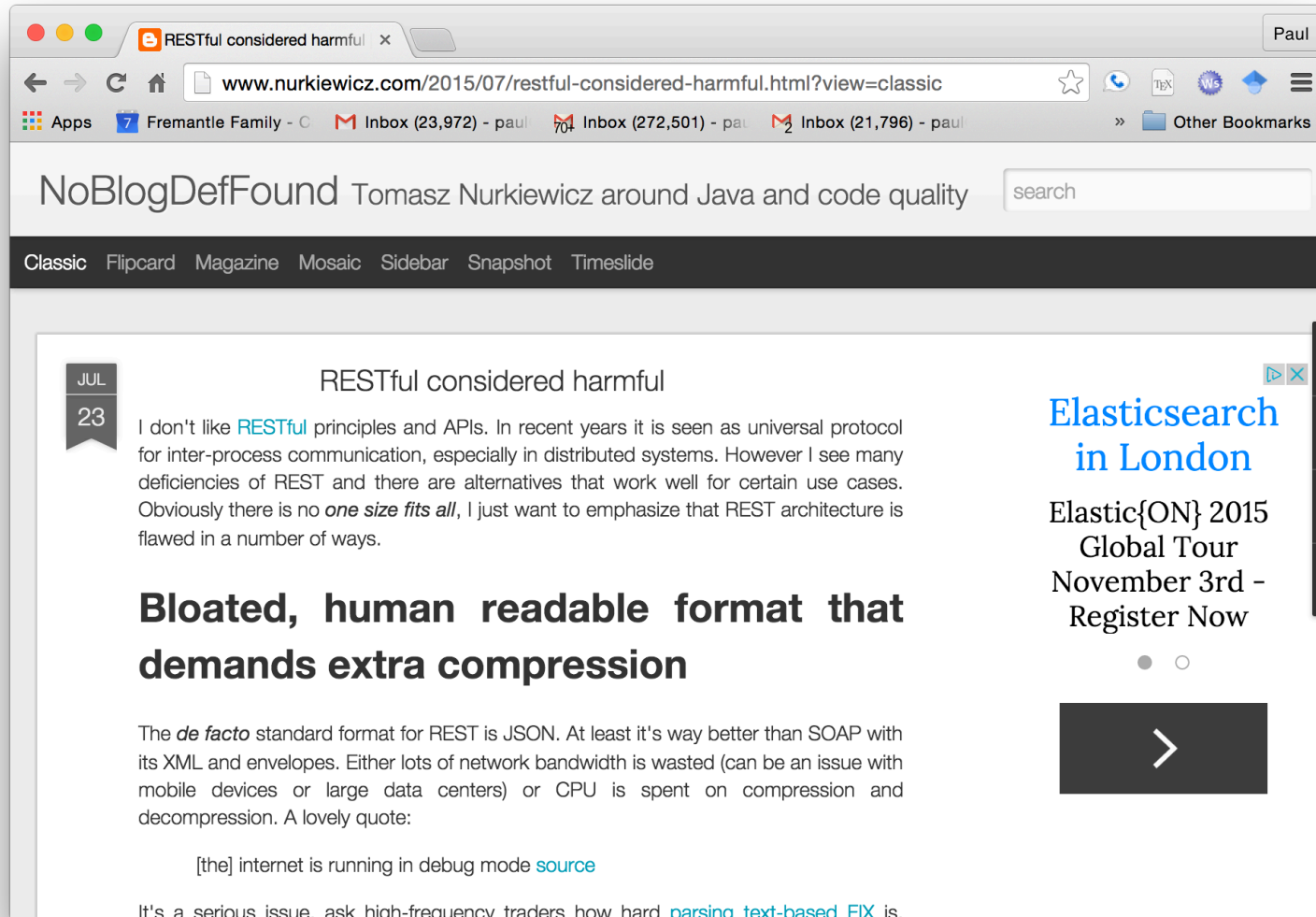
This document describes HTCPCP, a protocol for controlling, monitoring, and diagnosing coffee pots.

## 1. Rationale and Scope

There is coffee all over the world. Increasingly, in a world in which

# Not everyone agrees:

<http://www.nurkiewicz.com/2015/07/restful-considered-harmful.html>



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Anti-REST concerns

- Bloated formats (equally applies to SOAP)
- Neither Schema nor Contract
- APIs and discovery instead of clear published machine-readable documentation
- No inbuilt batching, paging, sorting, etc
- CRUD only
- HTTP Status codes mixed with business replies
- Temporal Coupling
- Not clear enough what is REST and what isn't!
- Backwards compatibility



# Why REST Keeps Me Up At Night

News, Mobile

May. 15 2012 By [Guest Author](#)



*This guest post comes from Daniel Jacobson ([@daniel\\_jacobson](#)), director of engineering for the [Netflix API](#). Prior to Netflix, Daniel ran application development for [NPR](#) where he created the [NPR API](#), among other things. He is also the co-author of [APIs: A Strategy Guide](#) and a frequent contributor to [ProgrammableWeb](#) and the [Netflix Tech Blog](#).*

With respect to Web APIs, the industry has clearly and emphatically landed on REST as the standard way to implement these services. And for good reason... REST, which is generally implemented as a one-size-fits-all solution, is an excellent choice for a most companies who wish to expose their content to third parties, mobile app developers, partners, internal teams, etc. There are many tomes about what REST is and how best to implement it, so I won't go into detail here. But if I were to sum up the value proposition to these companies of the traditional REST solution, I would describe it as:

*REST APIs are excellent at handling requests in a generic way, establishing a set of rules that allow a large number of known and unknown developers to easily consume the services that the API offers.*

<http://www.programmableweb.com/news/why-rest-keeps-me-night/2012/05/15>

# Questions?



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>