Approaching the Unacceptable Workload Boundary

Baron Schwartz • SREcon18 Americas

Logistics & Stuff

Slides are at xaprb.com/talks/.

Ask questions anytime.

Please get in touch: @xaprb or baron@vividcortex.com.

Introduction

What happens as systems get bigger and more heavily loaded?

Introduction

What happens as systems get bigger and more heavily loaded?

- What is a system's operating domain?

Introduction

What happens as systems get bigger and more heavily loaded?

- What is a system's operating domain?
- How is load defined?

Introduction

What happens as systems get bigger and more heavily loaded?

- What is a system's operating domain?
- How is load defined?
- Where is the load limit? How can you see it coming?

Introduction

What happens as systems get bigger and more heavily loaded?

- What is a system's operating domain?
- How is load defined?
- Where is the load limit? How can you see it coming?
- How does the system behave near this limit?

Introduction

What happens as systems get bigger and more heavily loaded?

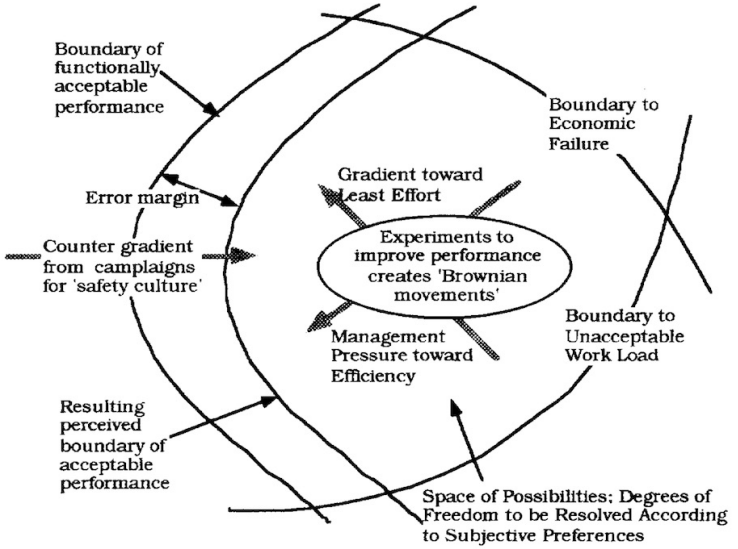
- What is a system's operating domain?
- How is load defined?
- Where is the load limit? How can you see it coming?
- How does the system behave near this limit?
- Can you measure and model this behavior?

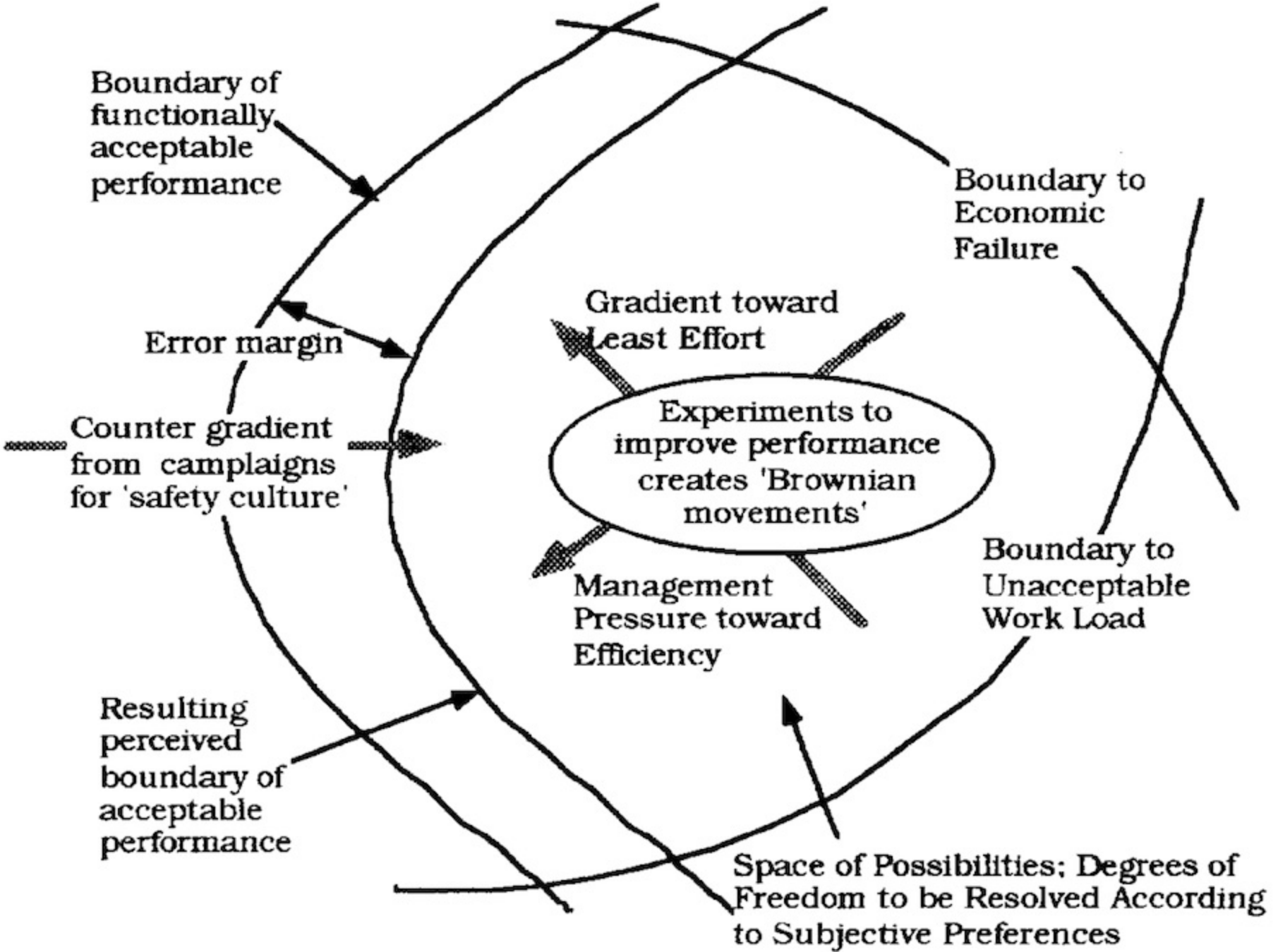
The Operating Domain

Operating Domain and Failure Boundaries

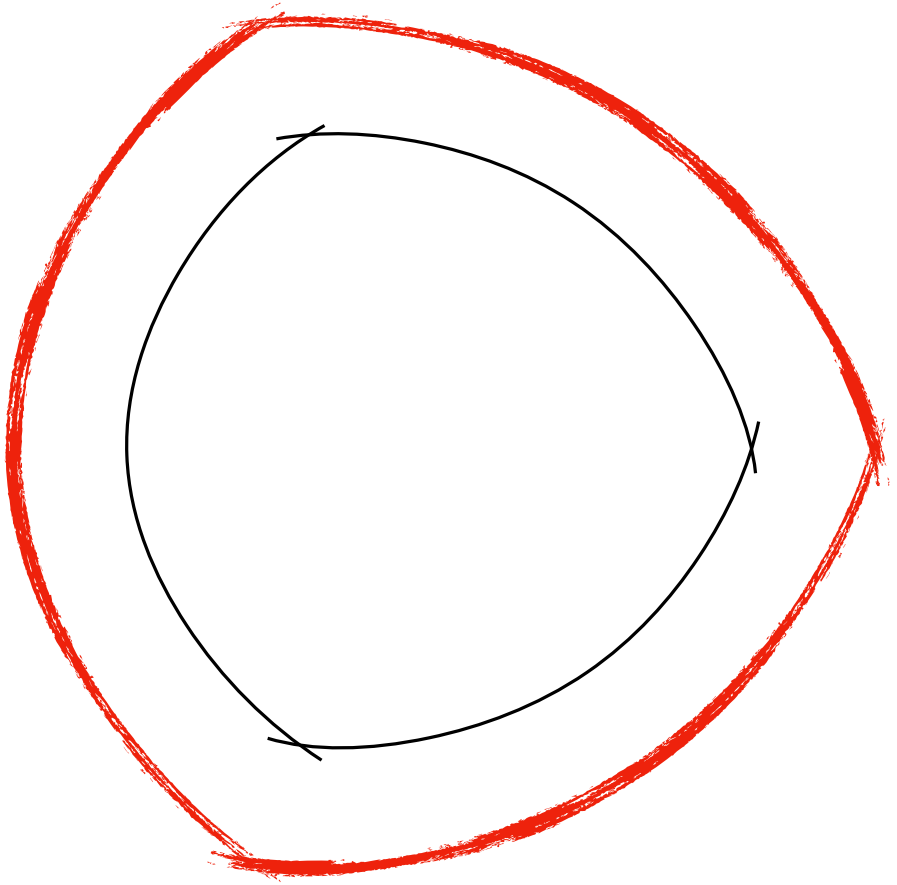
Rasmussen's model describes an **operating domain** bounded by economic risk, effort, and safety.

The system's **operating state** is a point within the domain, always moving around.

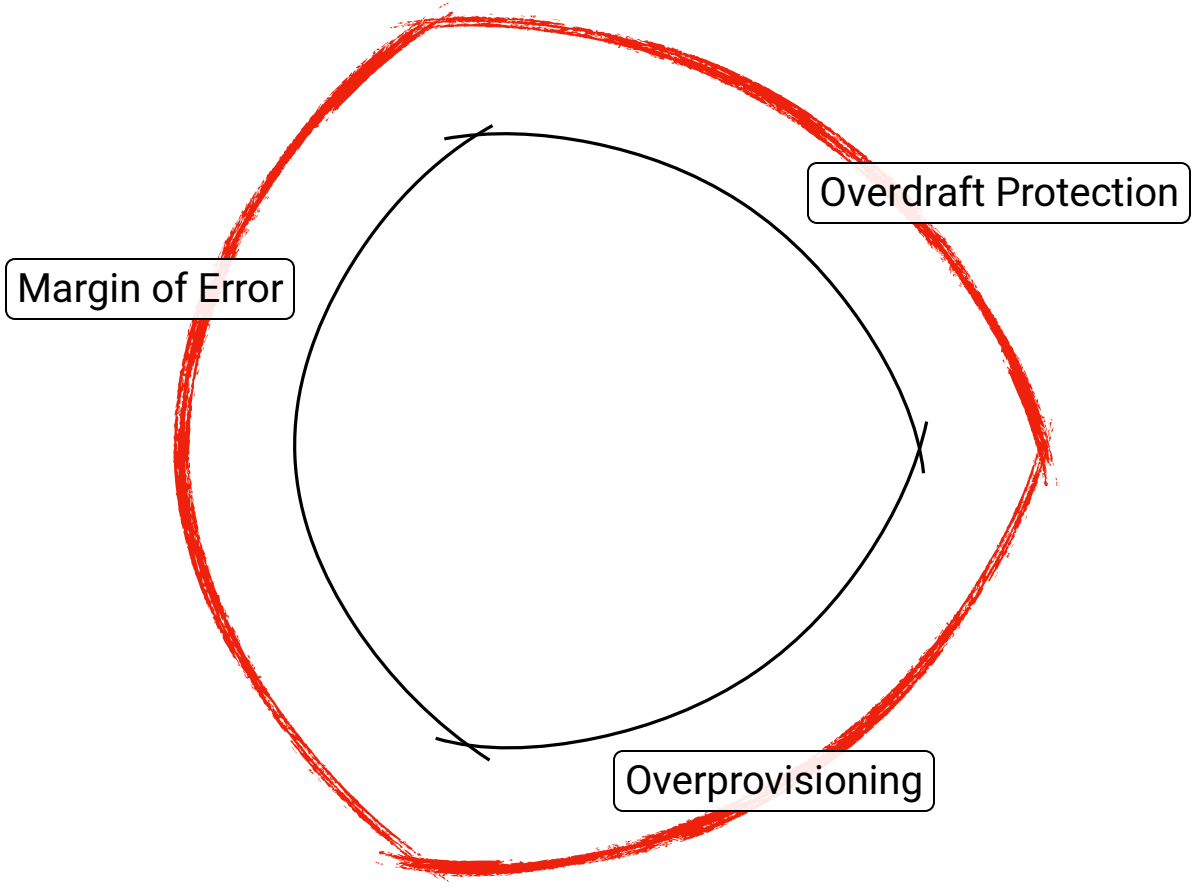



Boundary of functionally acceptable performance
Boundary to Economic Failure
Gradient toward Least Effort
Error margin
Experiments to improve performance creates 'Brownian movements'
Counter gradient from camplaigns for 'safety culture'
Management Pressure toward Efficiency
Boundary to Unacceptable Work Load
Resulting perceived boundary of acceptable performance
Space of Possibilities; Degrees of Freedom to be Resolved According to Subjective Preferences

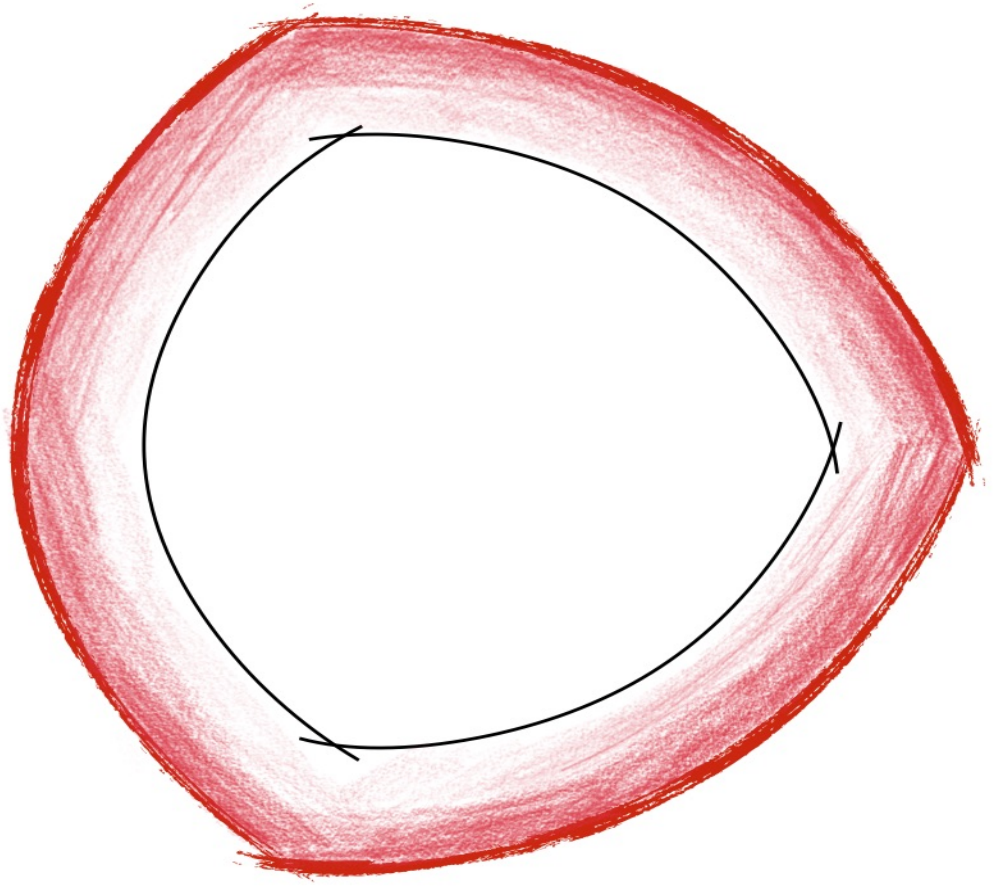
The Actual Boundaries Are Unknown



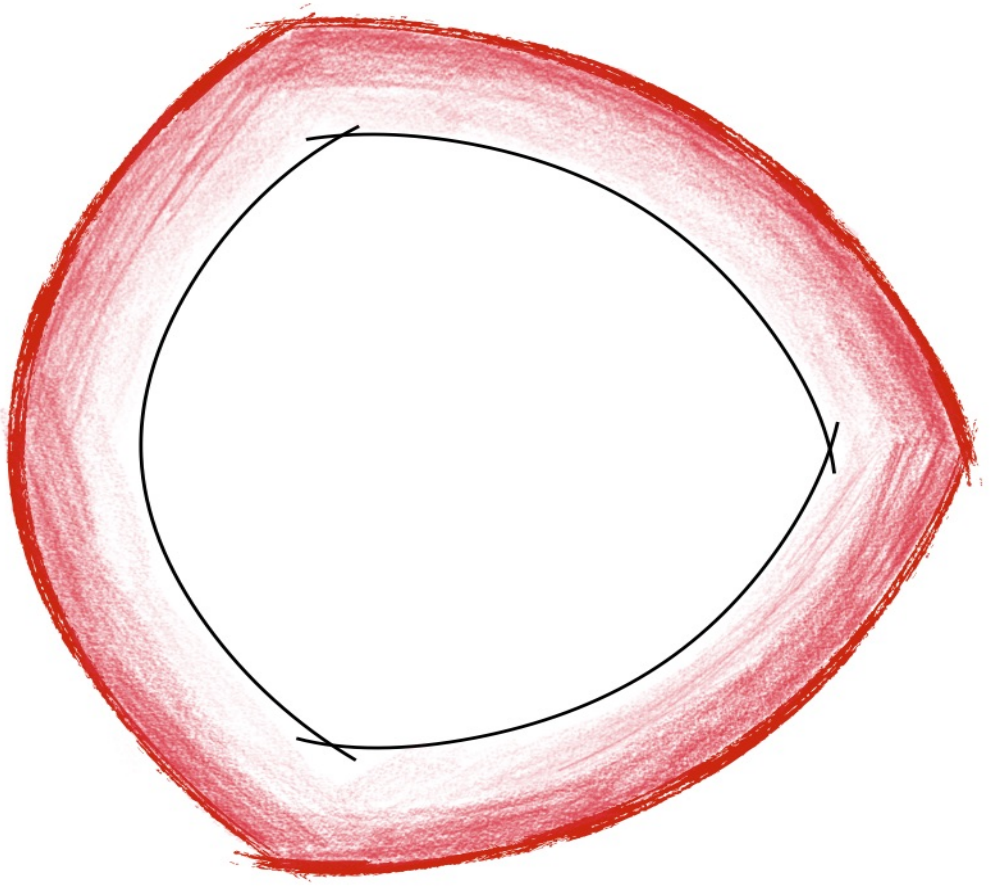
We Draw Limits Where We Think It's Safe



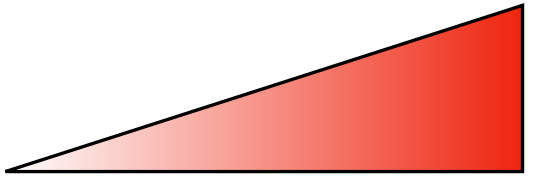
The Buffer Zone Is Nonlinear



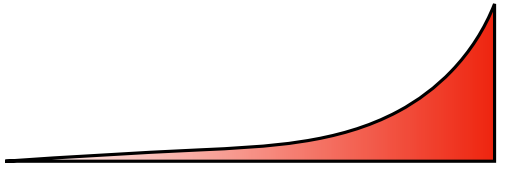
The Buffer Zone Is Nonlinear



We think the gradient looks like this.



It really looks more like this.



Complex Systems Run In Degraded Mode

Richard Cook lists 18 precepts of system failure in How Complex Systems Fail. Precepts 4) and 5) are especially relevant.

Complex Systems Run In Degraded Mode

Richard Cook lists 18 precepts of system failure in How Complex Systems Fail. Precepts 4) and 5) are especially relevant.

> **4) Complex systems contain changing mixtures of failures latent within them.** The complexity of these systems makes it impossible for them to run without multiple flaws being present.
>
> **5) Complex systems run in degraded mode.** A corollary to the preceding point is that complex systems run as broken systems.

System Load

What Is The Definition Of Load?

There's no one right answer to this question, but there's a **useful answer** for this discussion.

What Is The Definition Of Load?

There's no one right answer to this question, but there's a **useful answer** for this discussion.

Load is the **sum of task residence times** during an observation interval T. This is equivalent to average **concurrency** of tasks queued or in service:

$$N = \frac{\sum R}{T}$$

You can prove this with Little's Law.

Load, Utilization, And Queueing

Load (concurrency) is related to **utilization and queue length**, but it's not the same.

Load, Utilization, And Queueing

Load (concurrency) is related to **utilization and queue length**, but it's not the same.

- Concurrency is the number of requests in process simultaneously.

Load, Utilization, And Queueing

Load (concurrency) is related to **utilization and queue length**, but it's not the same.

- Concurrency is the number of requests in process simultaneously.
- Average concurrency is an average over an observation interval T.

Load, Utilization, And Queueing

Load (concurrency) is related to **utilization and queue length**, but it's not the same.

- Concurrency is the number of requests in process simultaneously.
- Average concurrency is an average over an observation interval T.
- Utilization is the fraction of T that was busy.

Load, Utilization, And Queueing

Load (concurrency) is related to **utilization and queue length**, but it's not the same.

- Concurrency is the number of requests in process simultaneously.
- Average concurrency is an average over an observation interval T.
- Utilization is the fraction of T that was busy.
- Queue length is the instantaneous or time-averaged number of tasks waiting to be serviced.

Utilization, Queue Length, & Concurrency

By Little's Law, utilization and queue length are **types of concurrency**.

- Utilization is the concurrency of in-service tasks.

Utilization, Queue Length, & Concurrency

By Little's Law, utilization and queue length are **types of concurrency**.

- Utilization is the concurrency of in-service tasks.
- Queue length is the concurrency of queued tasks.

What Is The Load Limit?

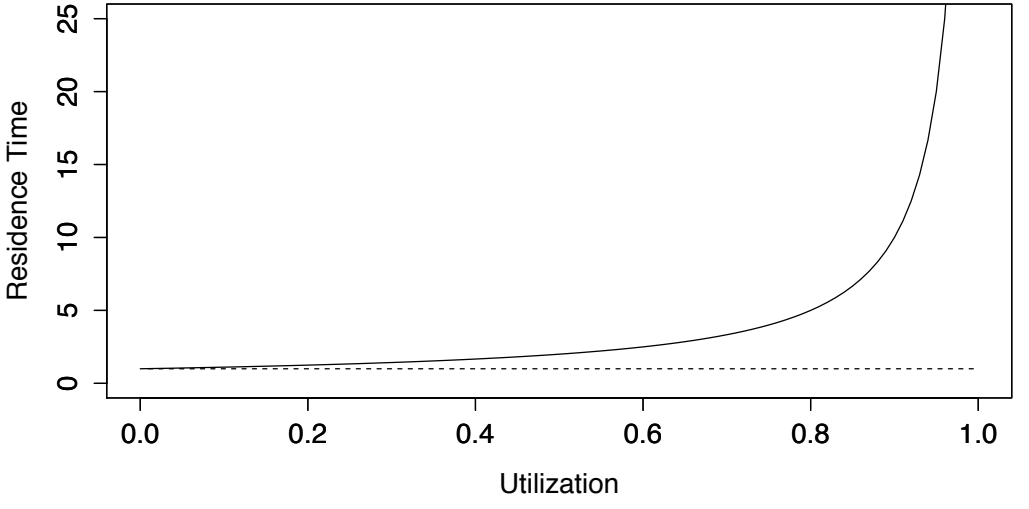
If the load limit were defined in terms of utilization, queueing theory could tell us where the **load limit** will be.

What Is The Load Limit?

If the load limit were defined in terms of utilization, queueing theory could tell us where the **load limit** will be. But it can't: load can be infinite, utilization ranges 0-1.

Plus it's impractical:

- The “hockey stick” queueing curve is hard to use
- The "knee" is unintuitive




Scalability

What's the Definition of Scalability?

There's a mathematical definition of scalability **as a function of concurrency**.

What's the Definition of Scalability?

There's a mathematical definition of scalability **as a function of concurrency**.

I'll illustrate it in terms of a **parallel processing system** that uses concurrency to achieve speedup.

Linear Scaling

Suppose a clustered system can complete **X tasks per second** with no parallelism.

Linear Scaling

Suppose a clustered system can complete **X tasks per second** with no parallelism.

With parallelism, it divides tasks and executes subtasks concurrently, **completing tasks faster**.

Linear Scaling

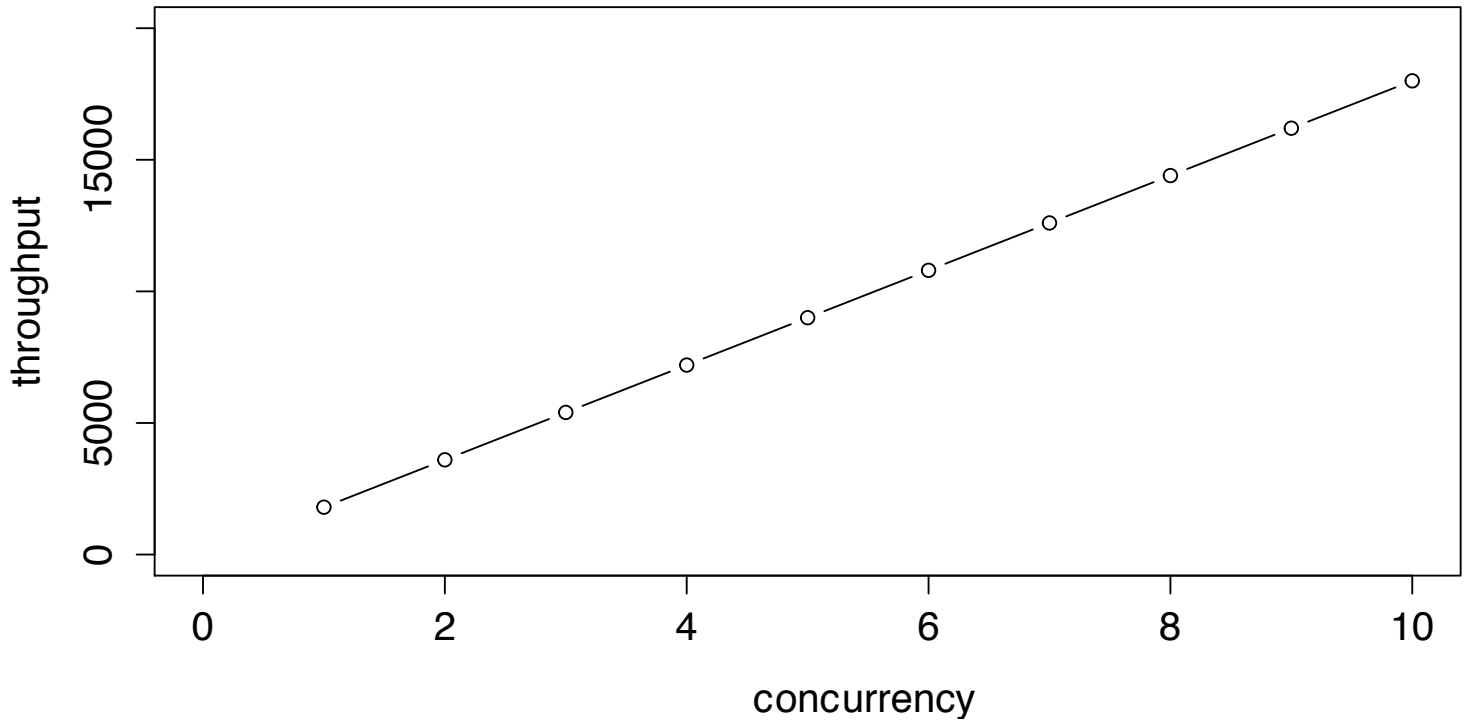
Suppose a clustered system can complete **X tasks per second** with no parallelism.

With parallelism, it divides tasks and executes subtasks concurrently, **completing tasks faster**.

Faster completion also means **increased throughput.**

Linear Scaling

Ideally, **throughput increases linearly with concurrency**.

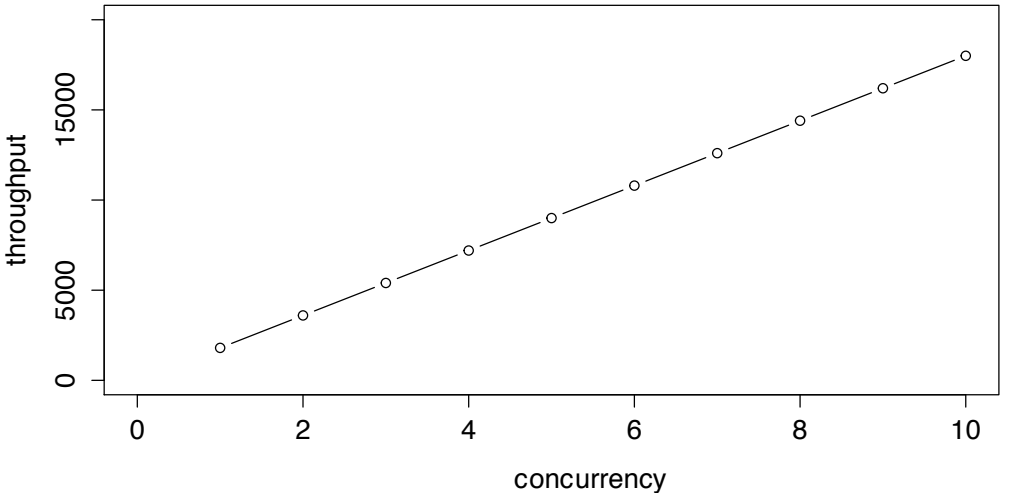


The Linear Scalability Equation

The equation that describes ideal scaling is

$$X(N) = \frac{\lambda N}{1}$$

where the slope is $\lambda = X(1)$.



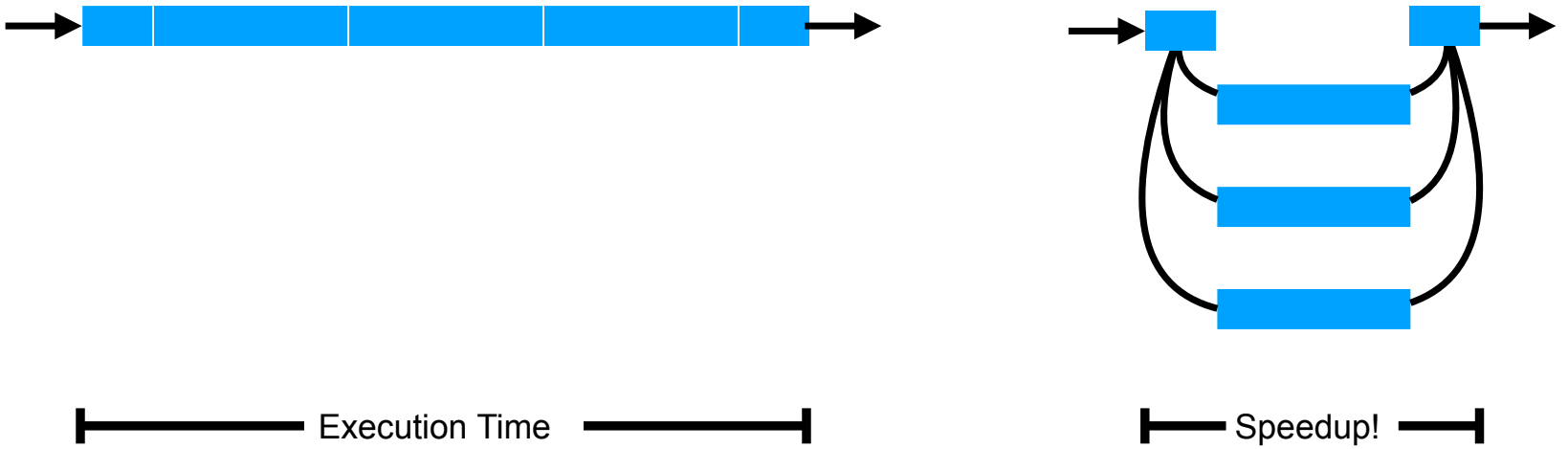
But Our Cluster Isn’t Perfect

Linear scaling comes from subdividing tasks **perfectly**.

But Our Cluster Isn’t Perfect

Linear scaling comes from subdividing tasks **perfectly**.

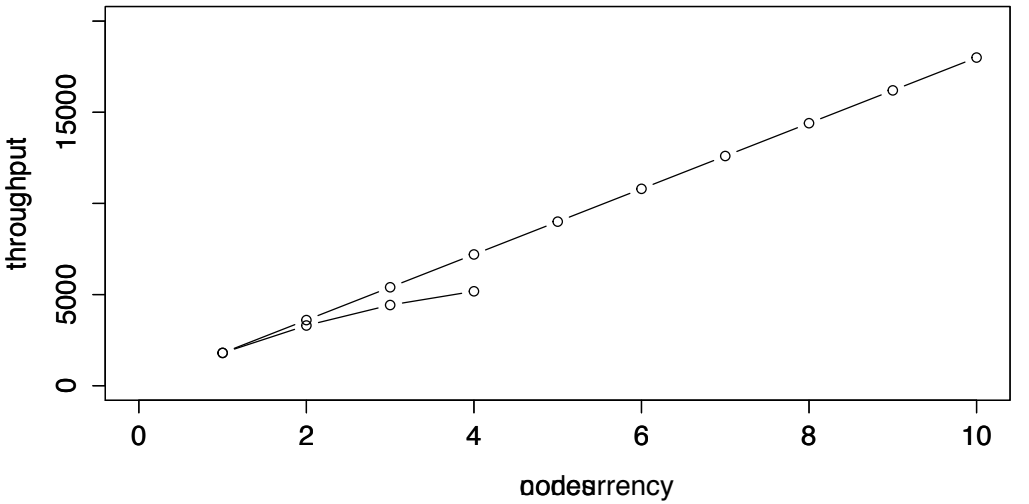
What if a portion isn’t subdividable?



Amdahl’s Law Describes Serialization

$$X(N) = \frac{\lambda N}{1 + \sigma(N-1)}$$

Amdahl's Law describes throughput when **a fraction σ can’t be parallelized**.



Amdahl's Law Has An Asymptote

$$X(N) = \frac{\lambda N}{1 + \sigma(N - 1)}$$

Parallelism delivers speedup, but there's a limit:

$$\lim_{N \to \infty} X(N) = \frac{1}{\sigma}$$

Amdahl's Law Has An Asymptote

$$X(N) = \frac{\lambda N}{1 + \sigma(N - 1)}$$

Parallelism delivers speedup, but there's a limit:

$$\lim_{N \to \infty} X(N) = \frac{1}{\sigma}$$

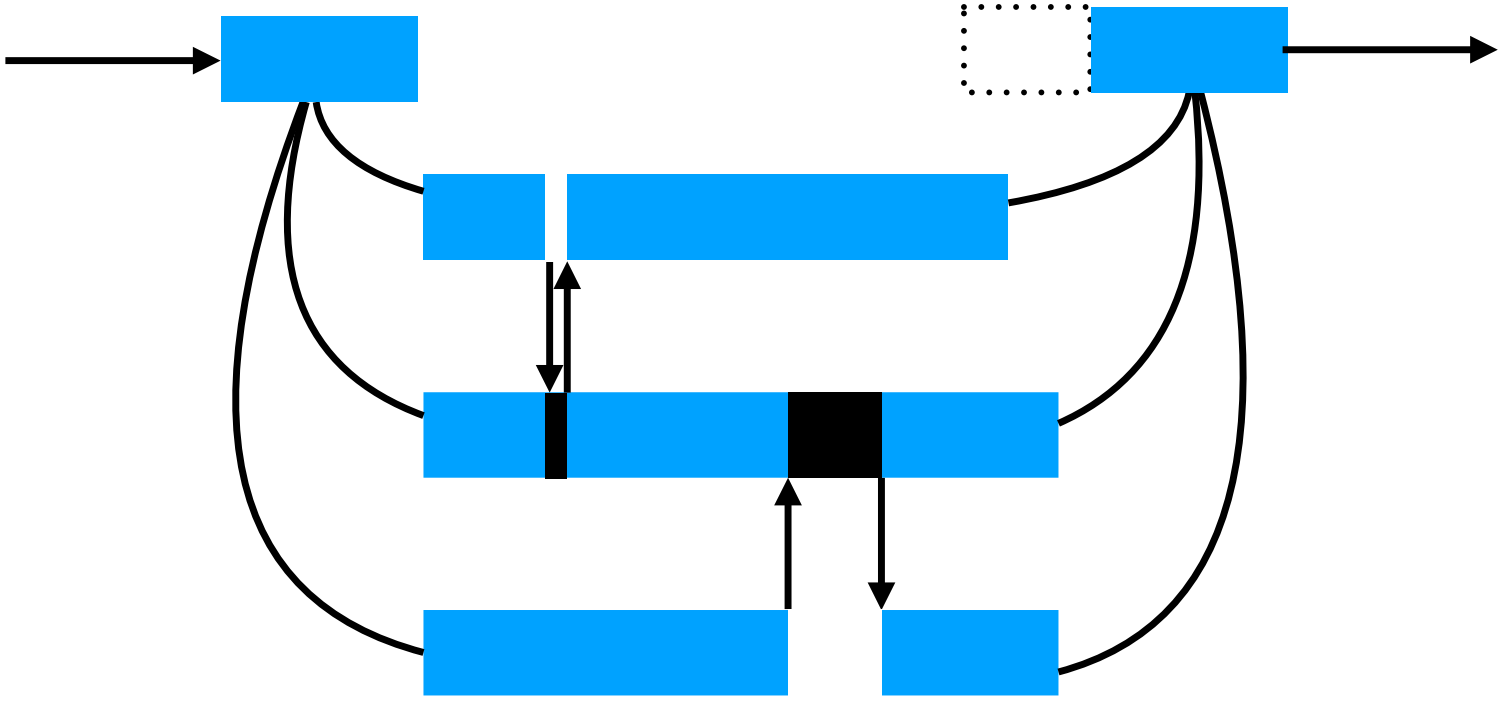
e.g. a 5% serialized task can't be sped up more than 20-fold.

What If Workers Coordinate?

Suppose the parallel workers **also have dependencies** on each other?

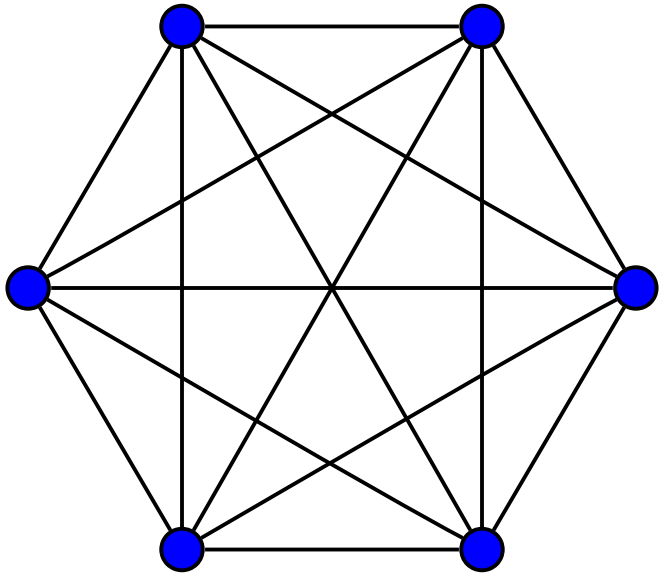
What If Workers Coordinate?

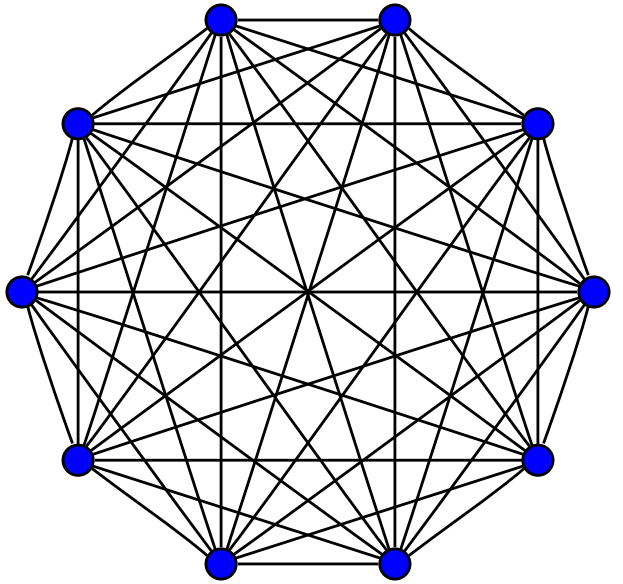
Suppose the parallel workers **also have dependencies** on each other?



How Bad Is Coordination?

N workers = $N(N-1)$ pairs of interactions, which is $\mathcal{O}(n^2)$ in N.



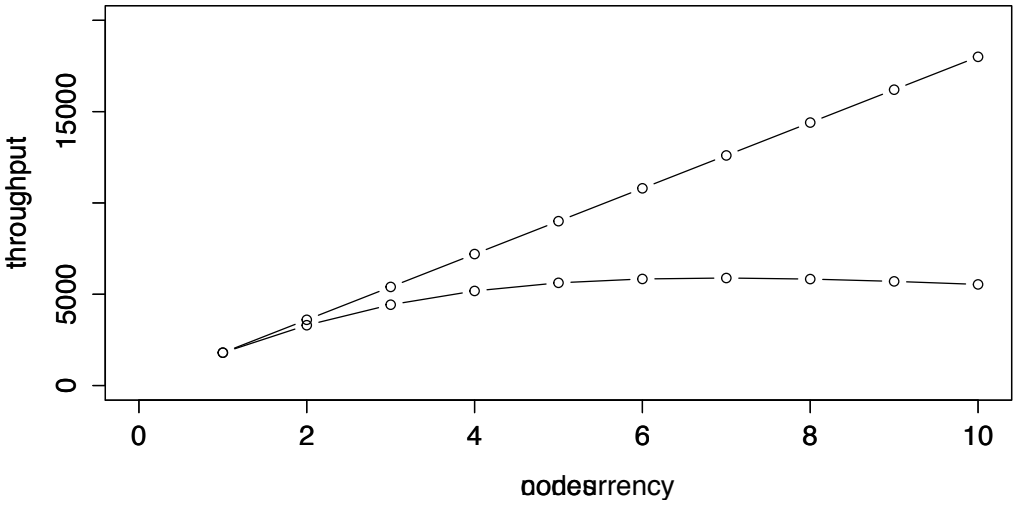


The Universal Scalability Law

$$X(N) = \frac{\lambda N}{1 + \sigma(N-1) + \kappa N(N-1)}$$

The USL adds a term for crosstalk, multiplied by the κ coefficient.

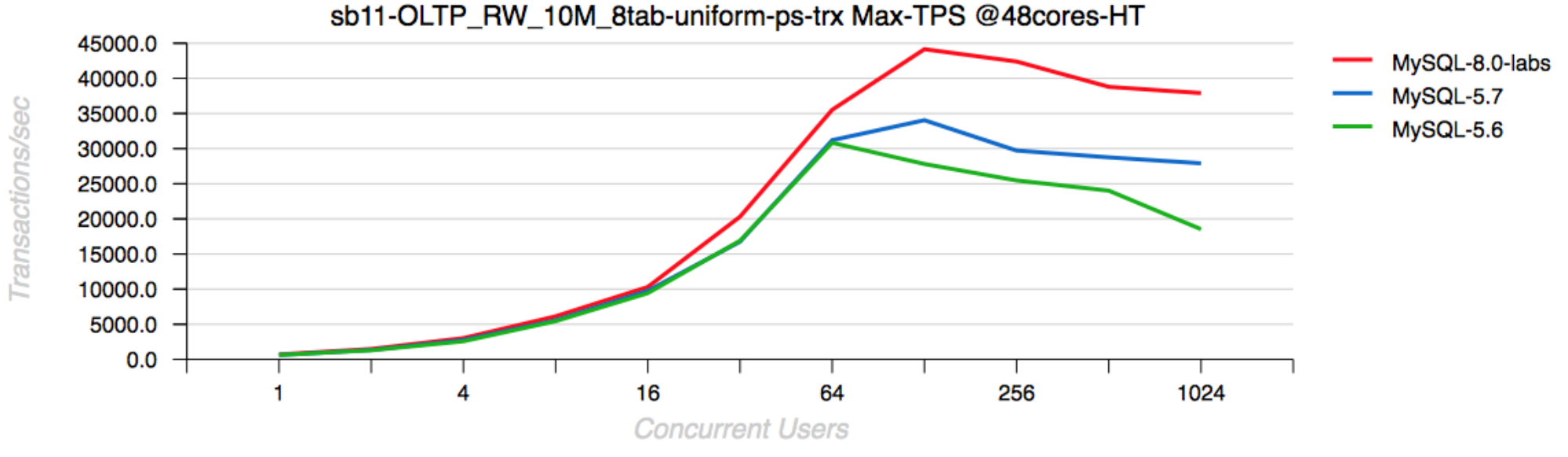
Now there's a **point of diminishing returns**!



Crosstalk is also called coordination or coherence.

You Already Know This

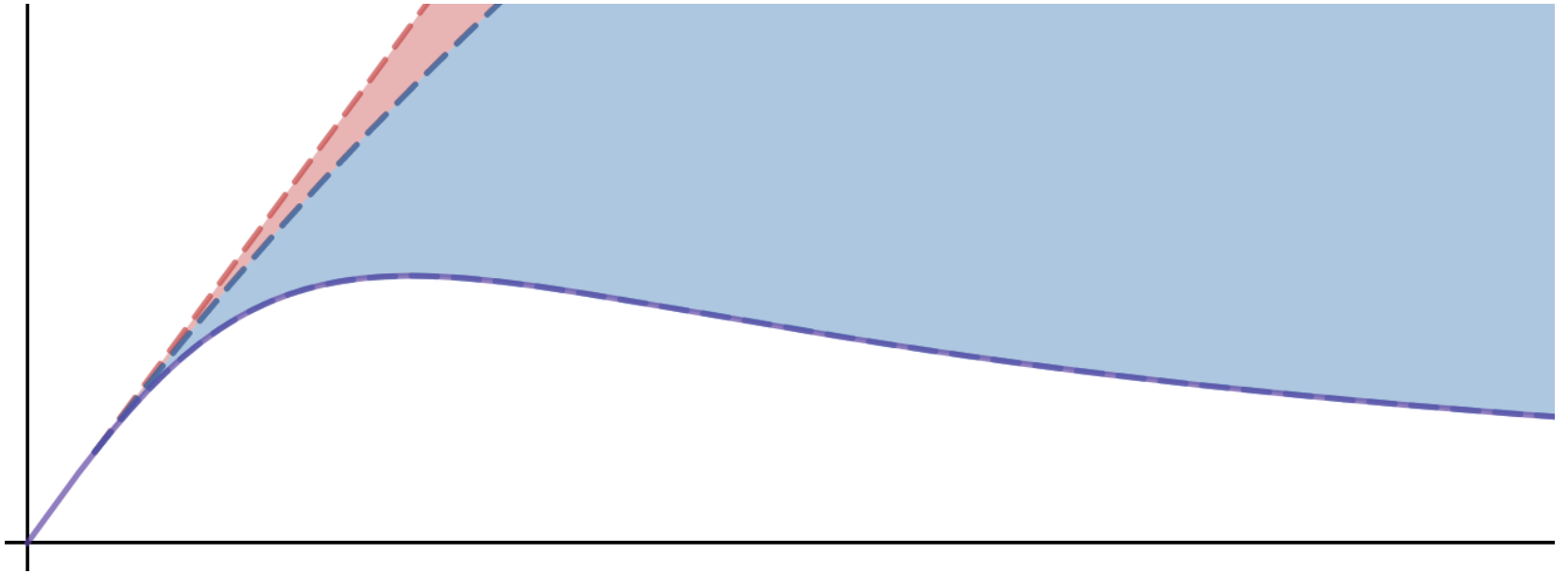
You've seen lots of benchmarks with diminishing returns.



Source: http://dimitrik.free.fr/blog/

The USL Describes Behavior Under Load

The USL explains the **highly nonlinear behavior** we know systems exhibit near their saturation point. desmos.com/calculator/3cycsgdl0b



A Summary Of The USL

The Universal Scalability Law defines **throughput as a function of concurrency**.

It explains how and why **systems don't scale linearly with load**.

What is the USL Good For?

Armed with the USL, you are ready to:

- Measure and model nonlinear behavior.
- Predict the onset of nonlinearity.
- Design better systems.

It's easy. Let's see how!


How To Measure, Model, And Predict

What To Measure

You can’t measure serialization & crosstalk directly.

What To Measure

You can’t measure serialization & crosstalk directly.

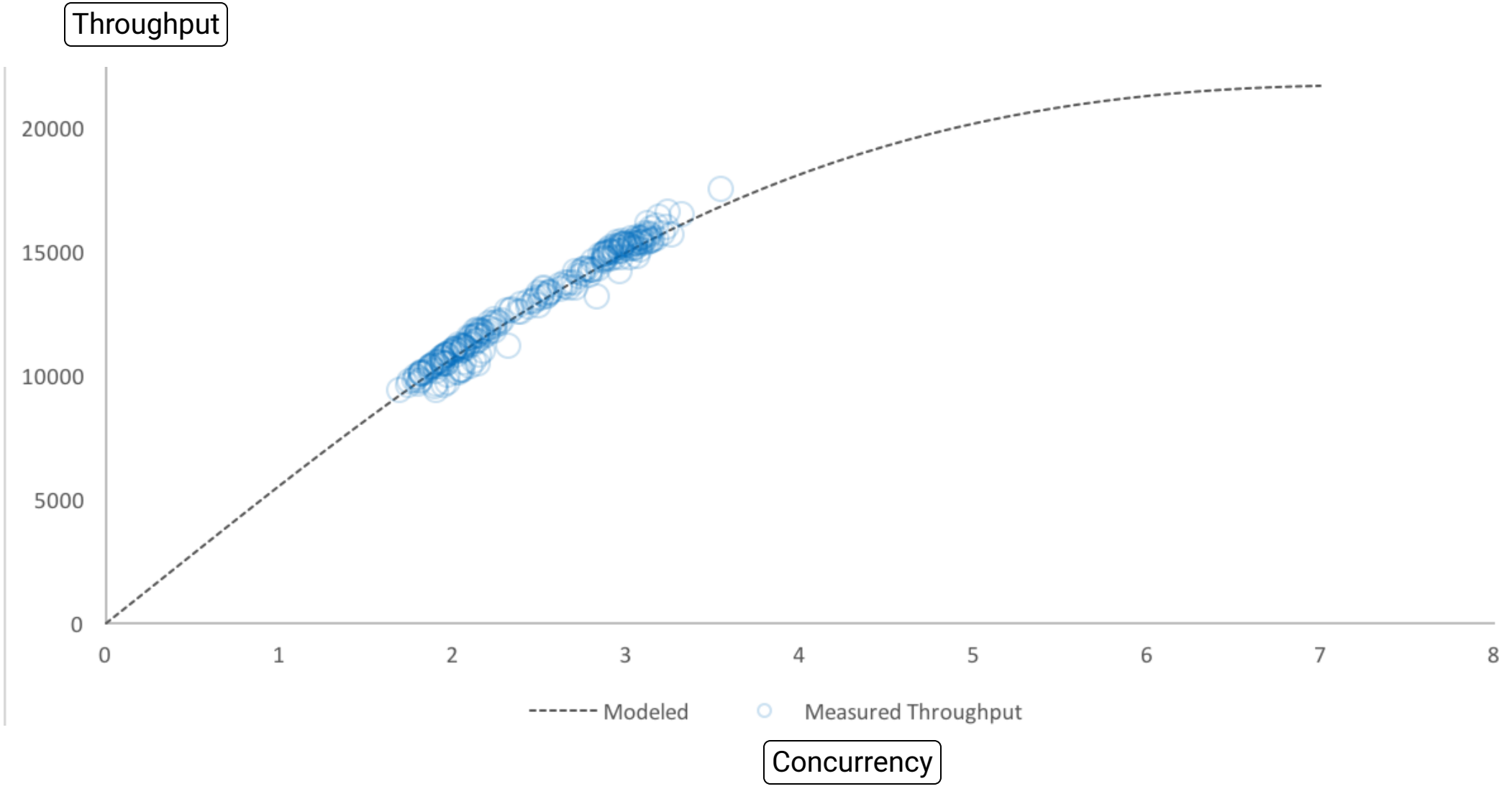
Instead, measure **throughput** and **concurrency**.

What To Measure

You can’t measure serialization & crosstalk directly.

Instead, measure **throughput** and **concurrency**.

Then **fit the USL model to the data** to estimate the parameters.


Throughput
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8
Modeled
Measured Throughput
Concurrency

How To Measure Concurrency, Pt. 1

Many systems have a metric of concurrency already. Look for a metric of **things actively working**.

- MySQL: `SHOW STATUS LIKE 'Threads_running'`
- Apache: active worker count

How To Measure Concurrency, Pt. 1

Many systems have a metric of concurrency already. Look for a metric of **things actively working**.

- MySQL: `SHOW STATUS LIKE 'Threads_running'`
- Apache: active worker count

It works well to **poll this e.g. 1x/sec**, then average these into 1- or 3-minute averages.

How To Measure Concurrency, Pt. 2

If there's no metric of concurrency, you can **sum up latencies and divide by the duration**.

$$N = \frac{\sum R}{T}$$

Plot Your Data

Simply **scatterplot your data** and eyeball it for sanity.

Plug The Data Into The Model

Paste the data into the Excel model I built.

Interpreting The Results

What does the output mean?

- Shows whether your system has **more serialization or crosstalk**.

Interpreting The Results

What does the output mean?

- Shows whether your system has **more serialization or crosstalk**.
- Shows the **estimated max load** where it'll stop scaling.

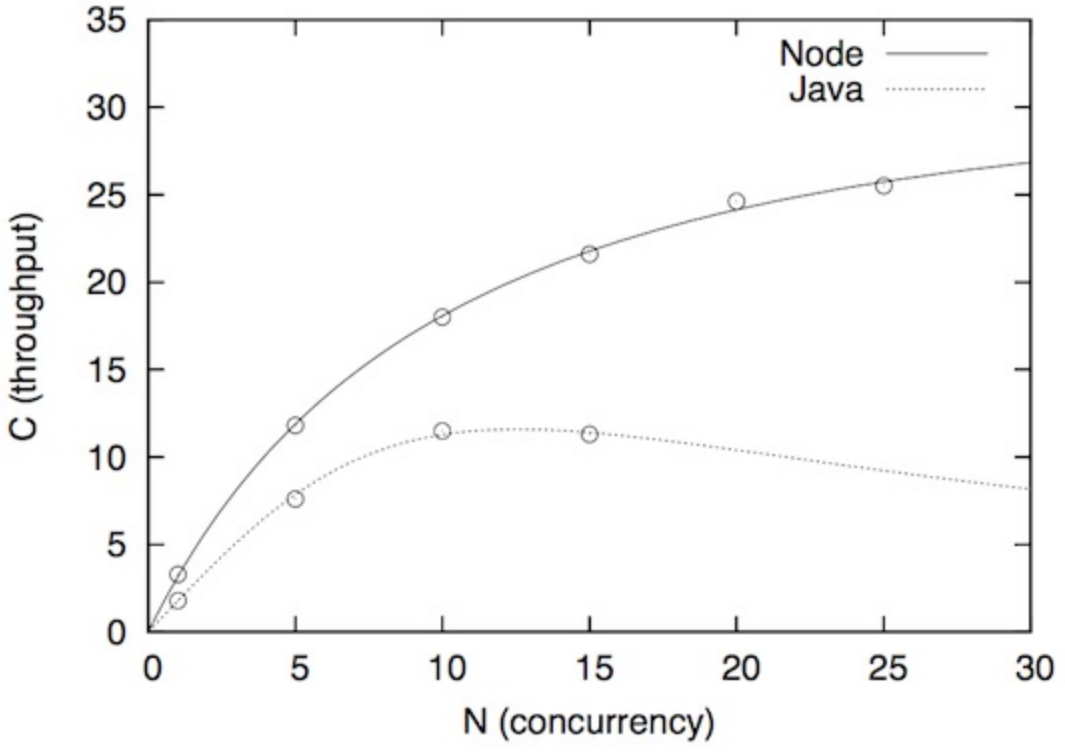
Interpreting The Results

What does the output mean?

- Shows whether your system has **more serialization or crosstalk**.
- Shows the **estimated max load** where it'll stop scaling.
- Helps you **predict nonlinearity**.

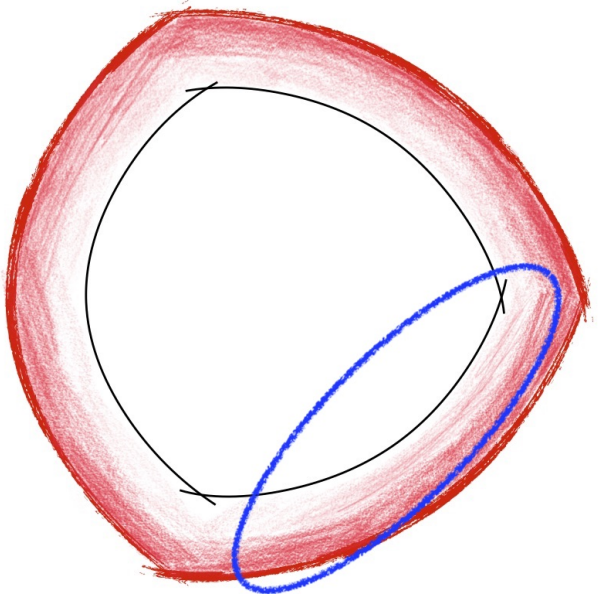
Paypal's NodeJS vs Java Benchmarks

Paypal's NodeJS vs Java benchmarks are a good example!



Bringing It Back To The Operating Domain

The USL is one way to understand **what happens near this boundary**.



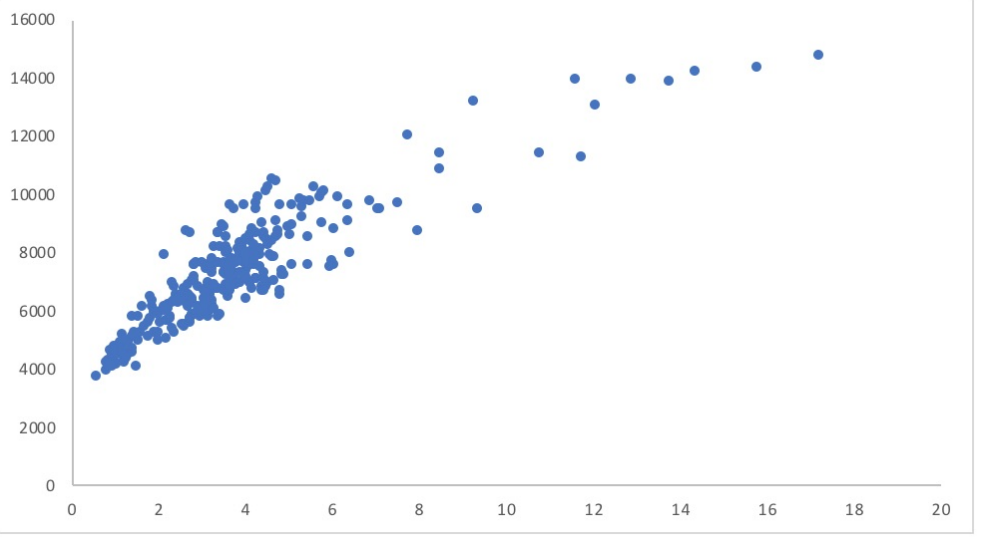
What Happens Here?

- When the system approaches **workload limits** it gets twitchy.
- You may be able to **see this approaching** before it gets bad.
- Simply scatterplotting **throughput vs concurrency** is super useful!

You Don't Need To Do Any Modeling!

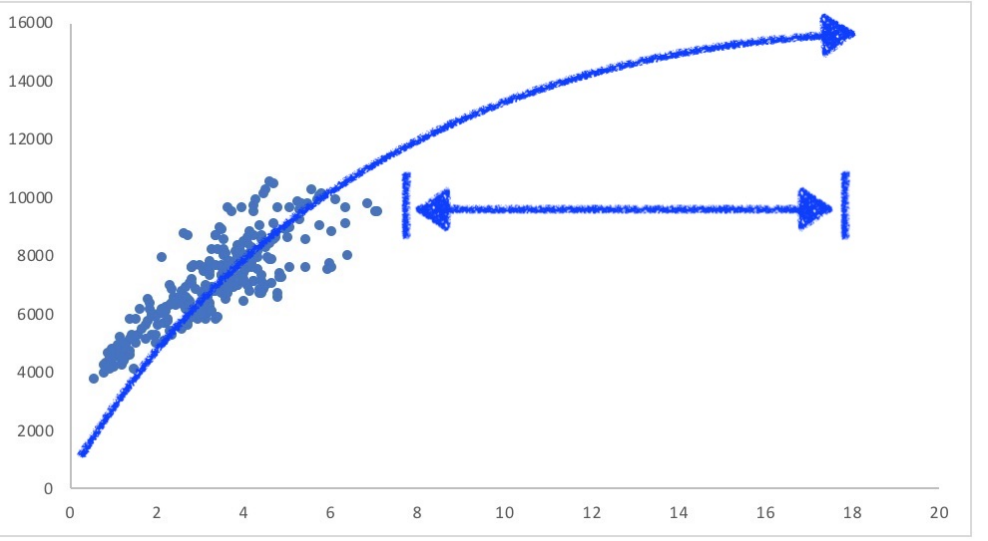
Let's take another look at this data.

What jumps out?



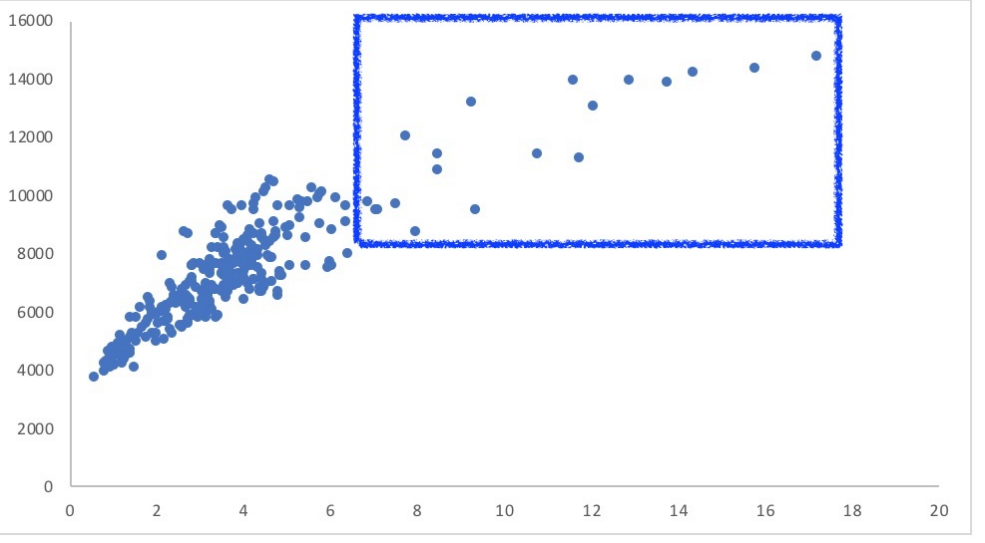
What If You Had Only The First Part?

- I would model and project out to the right.
- I'd see "hmm, it's **leveling off**."
- I'd say "don't count on much more than you see now."



Think Differently About Outlying Points

- Given all the data, I mentally cluster it into two parts.
- If the high-end outliers deviate, **it's nonlinear already.**
- Those points are evidence that the system is struggling there.
- You don't need to model anything to see that.



Some Resources

I wrote a book.

I created an Excel workbook.

These slides are at xaprb.com/talks.

Rasmussen's Model.

Richard Cook's talk about Resilience in Complex Adaptive Systems.