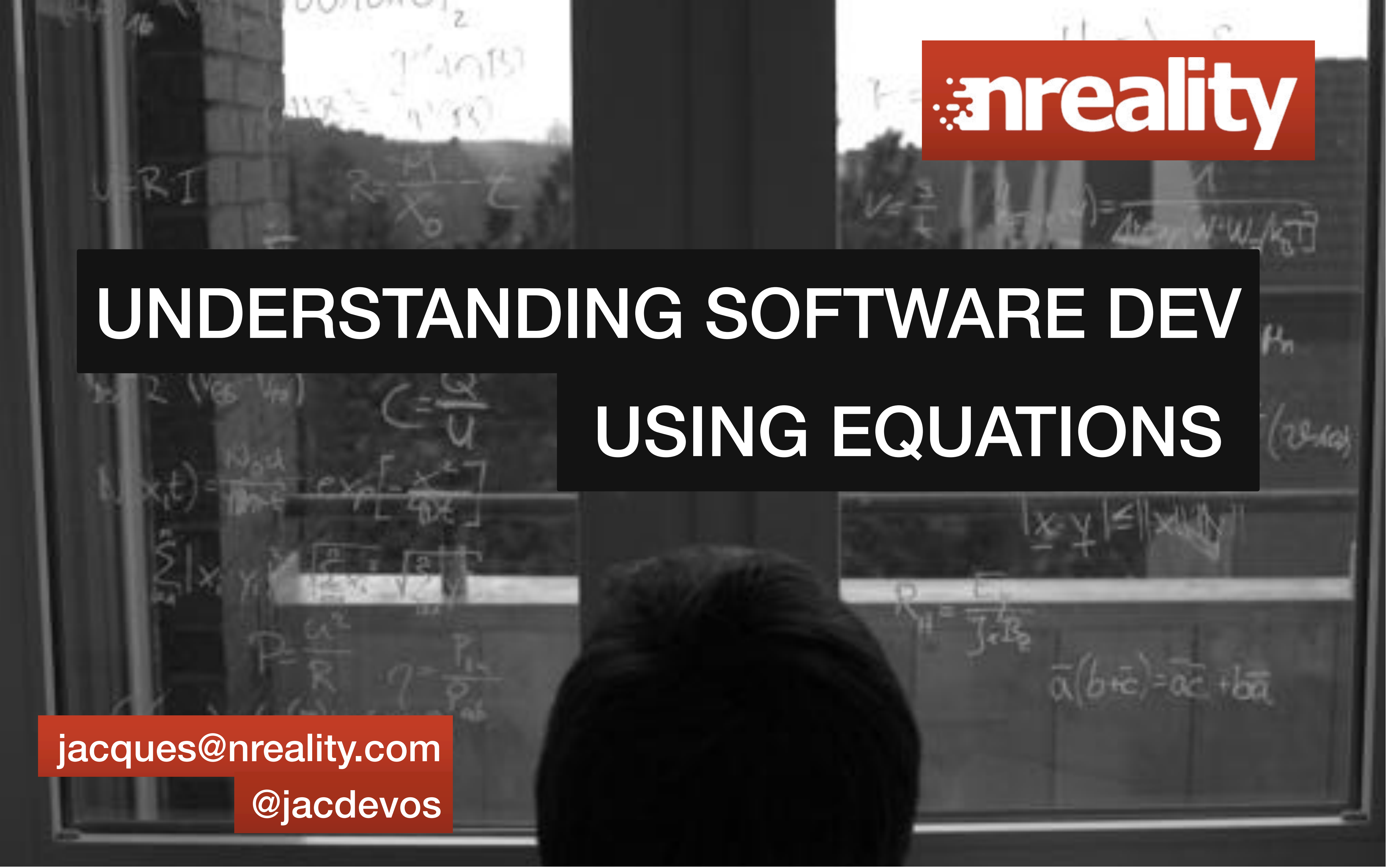


nreality

UNDERSTANDING SOFTWARE DEV USING EQUATIONS

jacques@nreality.com
@jacdevos

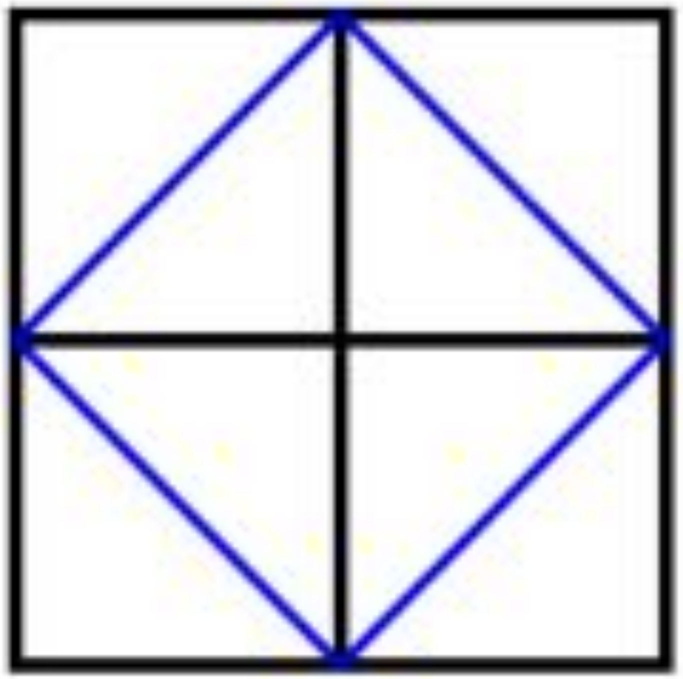
Software engineering is:

- Hard!
- Non-intuitive
- Misunderstood
- Seen as black-art done by amateurs

Typical problems

- Congestion and Dependencies
- Technical Debt
- Firefighting

How can math help us?



Socrates’ Meno

Clear thinking

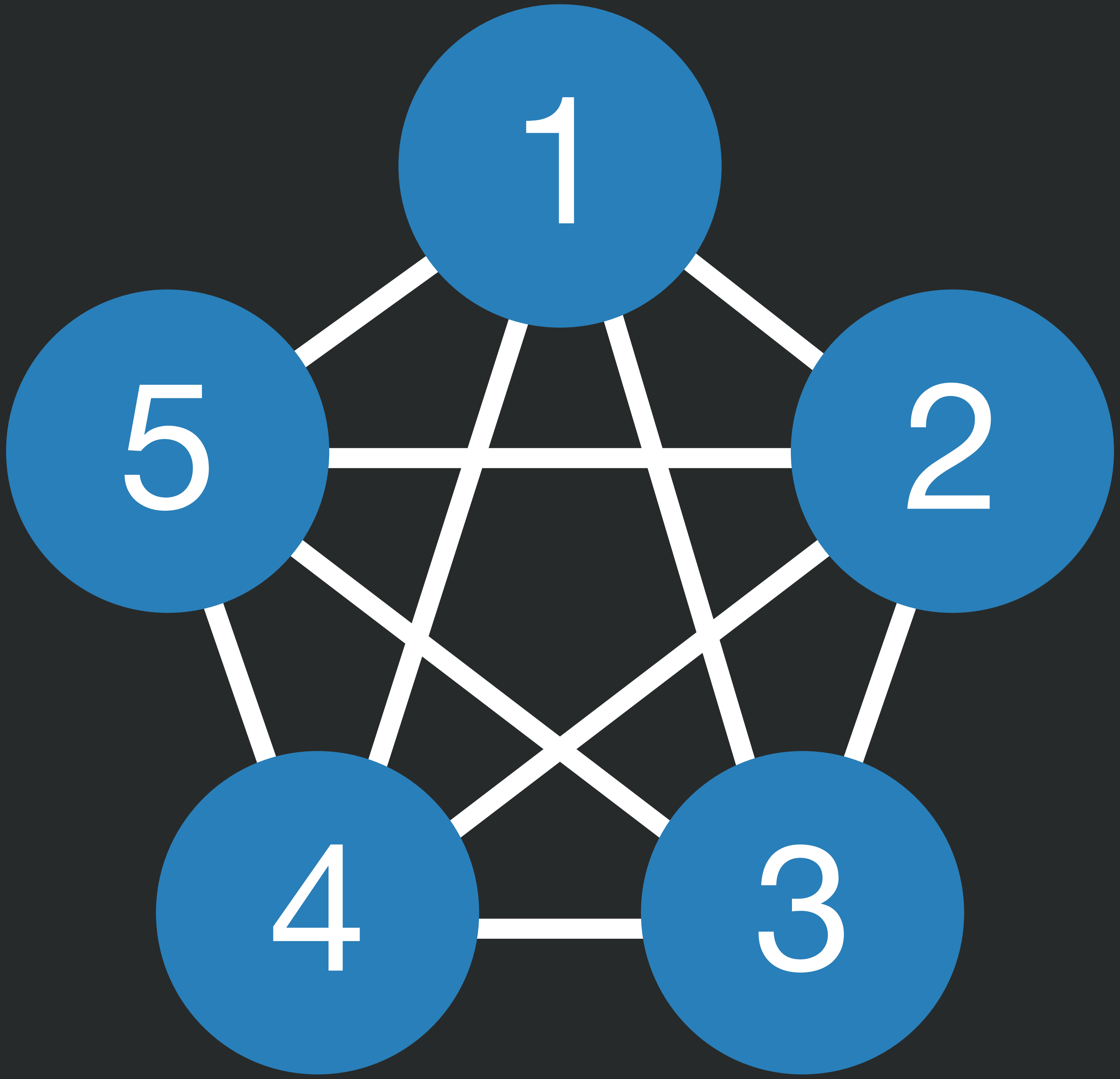
rather than precise proofs

Why do systems become so complicated?

http://complexitylabs.io/

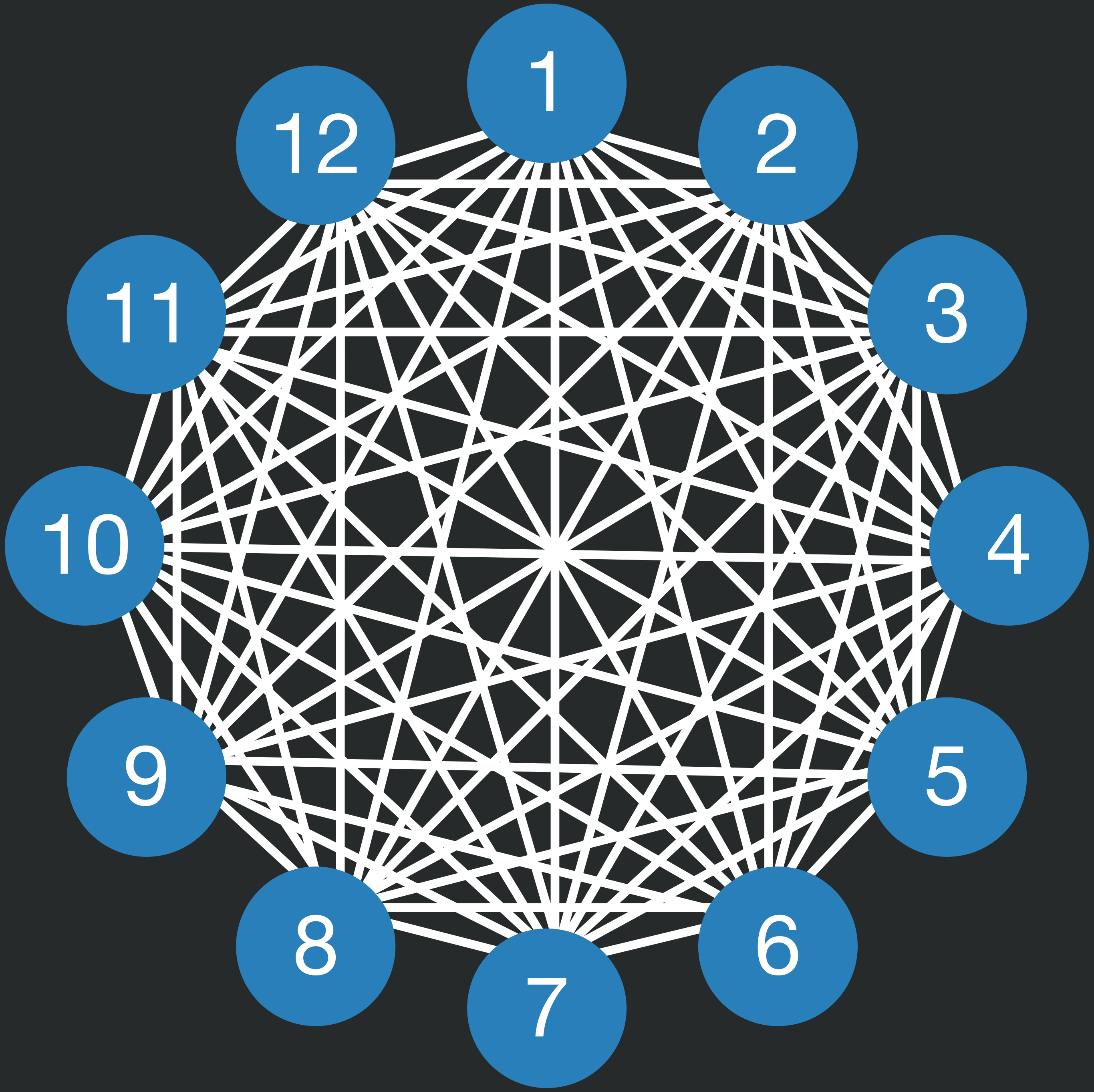
Understand complexity with Metcalfe's Law

$$\frac{n(n-1)}{2}$$



10

$$\frac{n(n-1)}{2}$$

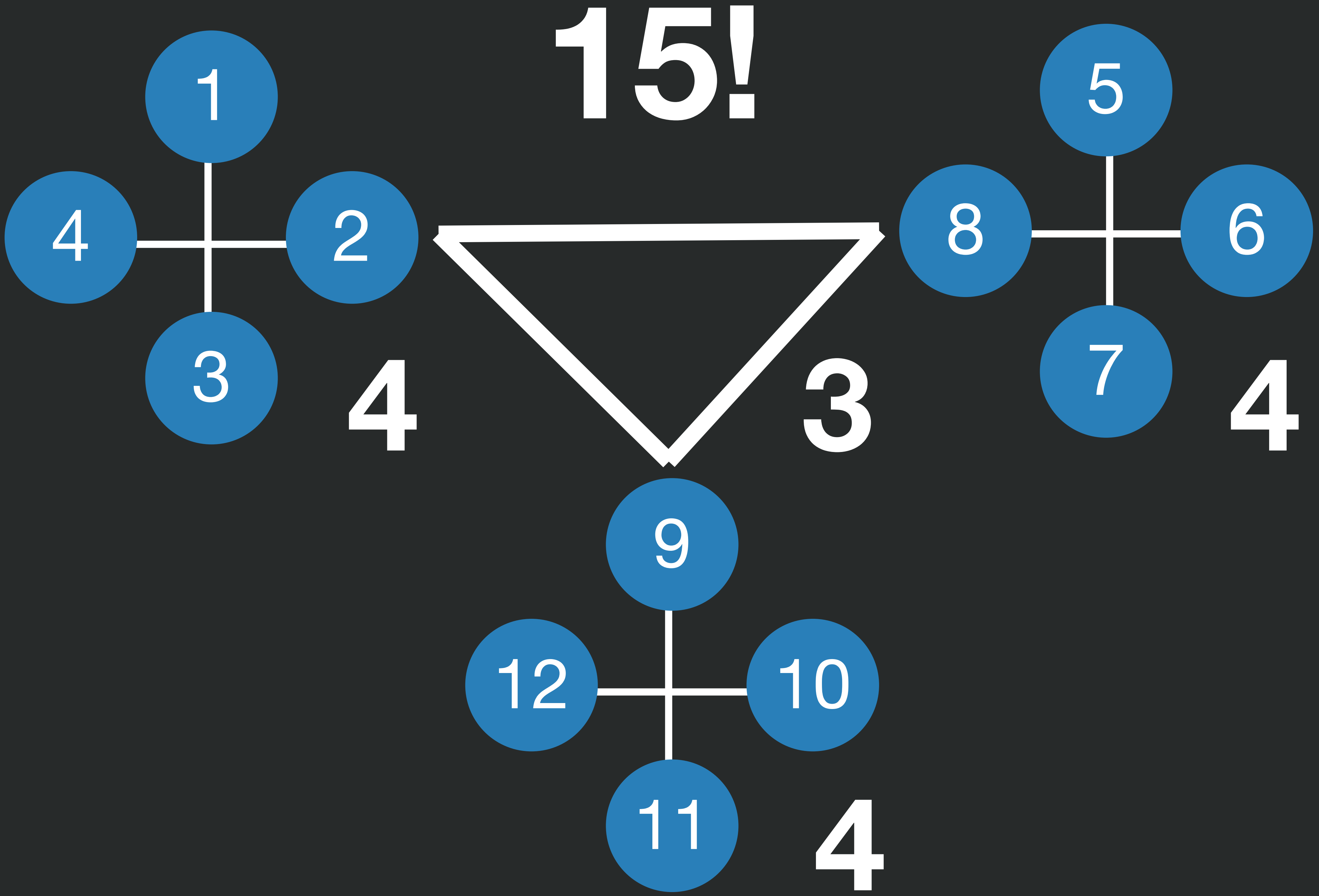


66

Complexity is caused by: quadratic increase in connections

How do we reduce complexity?

Design reduces dependencies


15!
1
4
2
3
4
3
5
8
6
7
4
9
12
10
11
4

Org structure
or
Code structure?

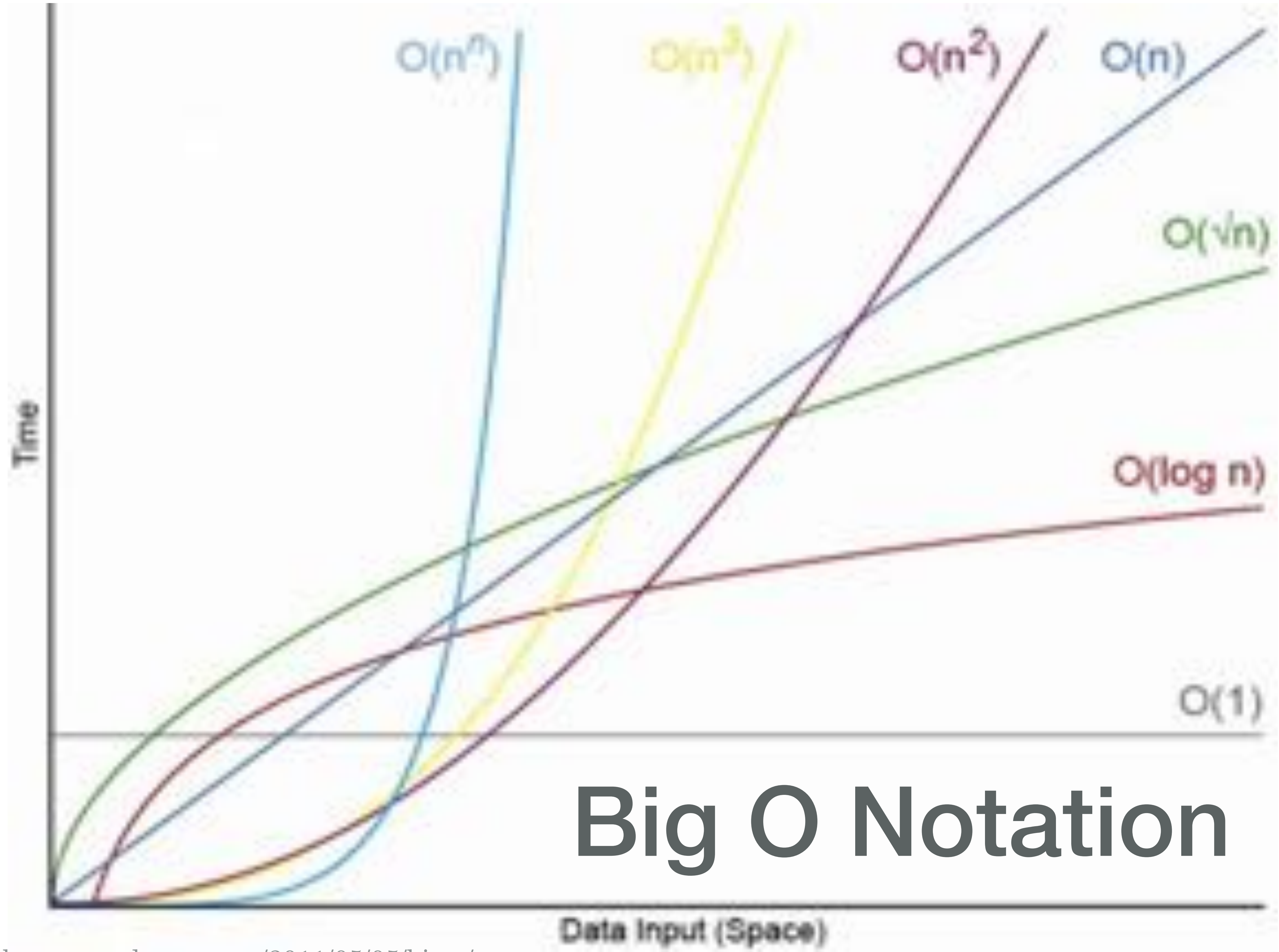
Summary:

Reduce connections to increase simplicity.

Remember Metcalfe’s Law!

But what about complexity over *TIME*?

Big O Notation

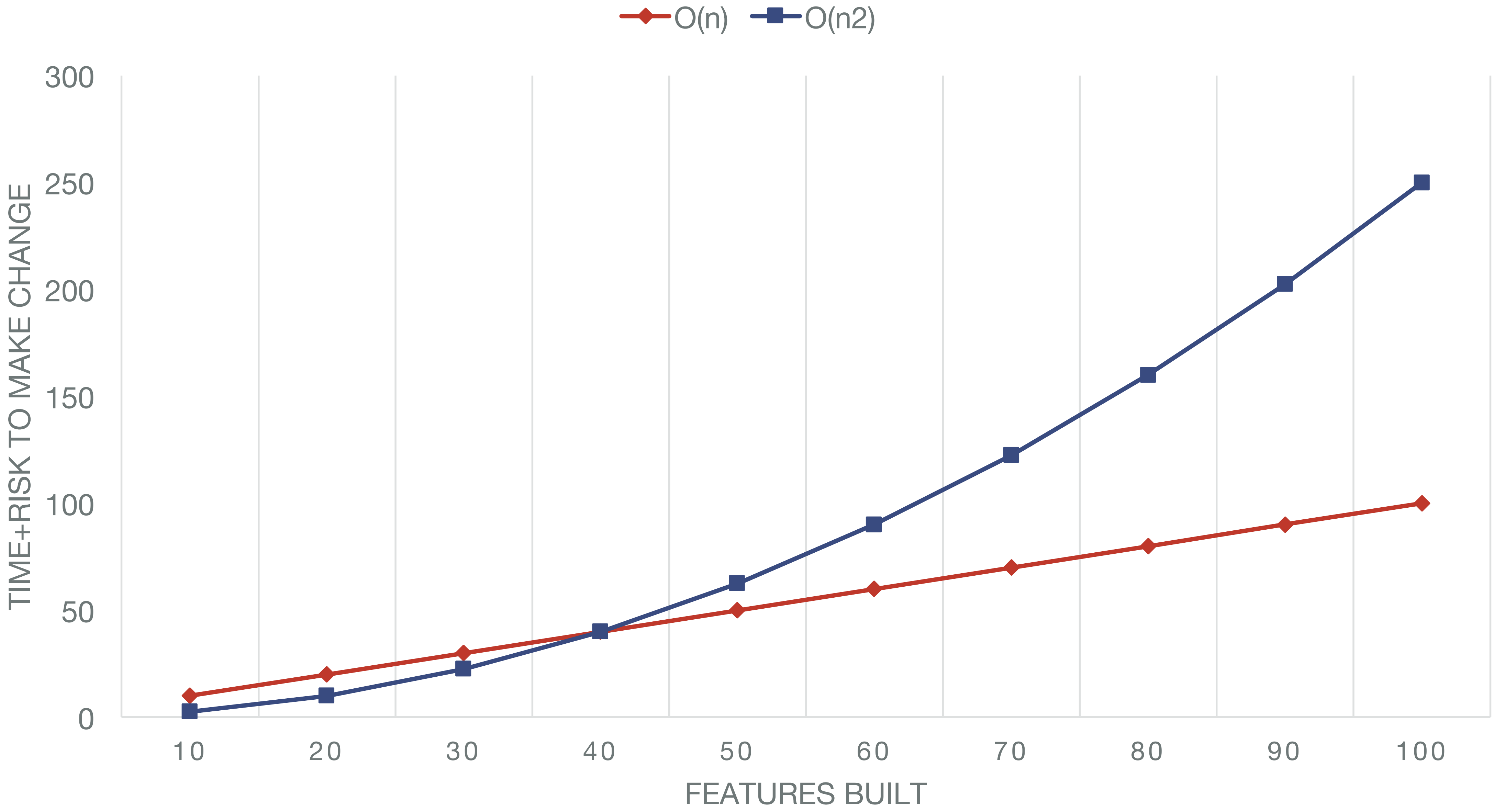


https://apelbaum.wordpress.com/2011/05/05/big-o/

Go slow to go fast with the right work algorithm

Big O

CODE CHANGE COMPLEXITY

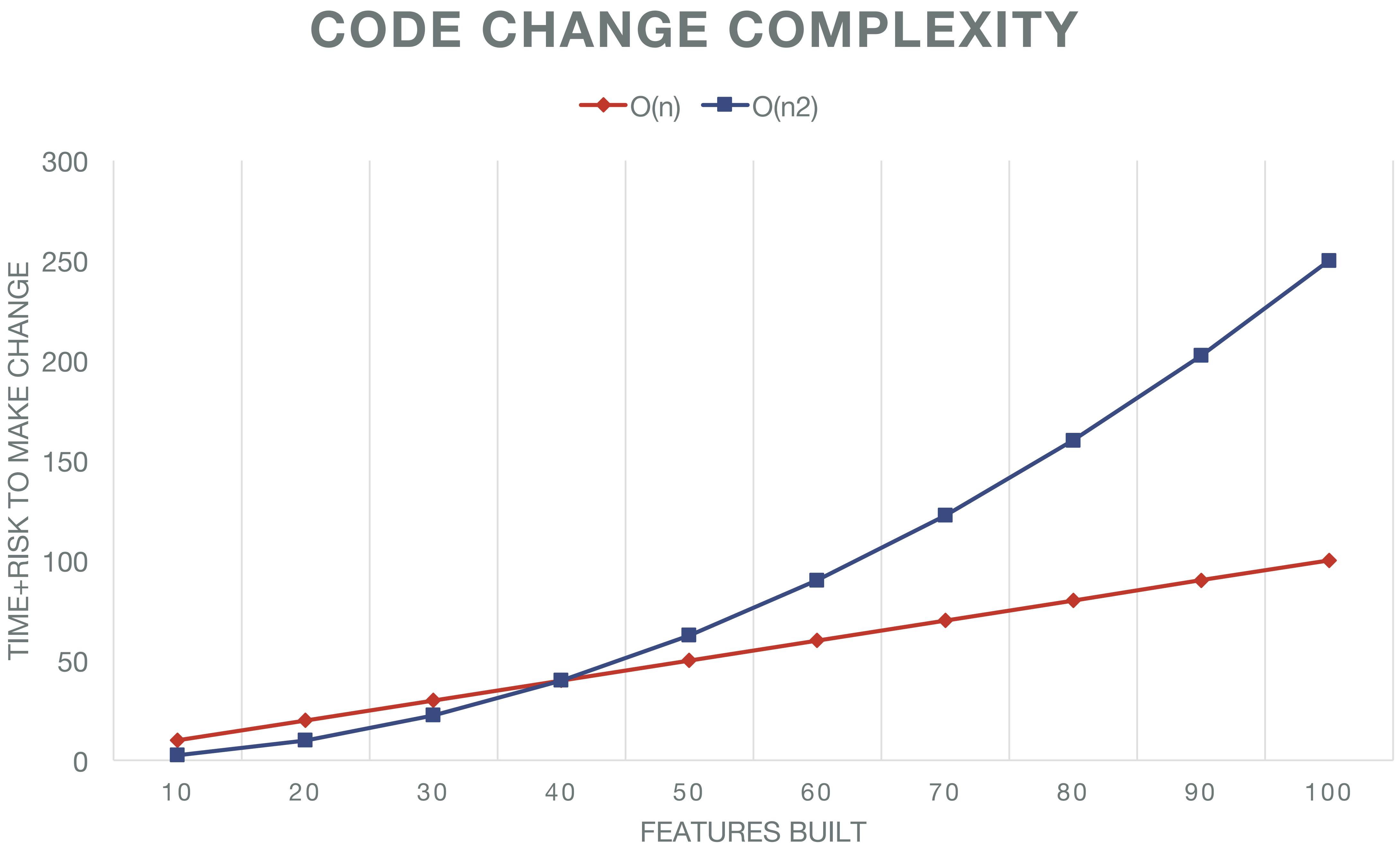


Clean Code gives $\boldsymbol{O(n)}$

- Easy to read
- SOLID design
- Boy scout refactoring
- Self-testing
- TDD

Messy code gives $\boldsymbol{O}(\boldsymbol{n}^2)$

- Manual regression testing
- Spaghetti dependencies
- Hart to understand


CODE CHANGE COMPLEXITY
O(n)
O(n2)
300
250
200
150
100
50
0
TIME+RISK TO MAKE CHANGE
10
20
30
40
50
60
70
80
90
100
FEATURES BUILT

Your algorithm of work determines order of time complexity

In summary

Clean code decreases dev cost over time. Go slow to go fast.

Think about work your algorithm with Big O!

We’re so busy! Why don’t we get stuff done?

We’re so busy! Why don’t we get stuff done?

Credit: http://www.autoevolution.com/

SHOCKWAVE TRAFFIC JAMS
RECREATED FOR FIRST TIME

Footage courtesy of
University of Nagoya,
Nagoya, Japan

youtu.be/Suugn-p5C1M

Maximise efficiency with Queueing Theory

Little’s Law

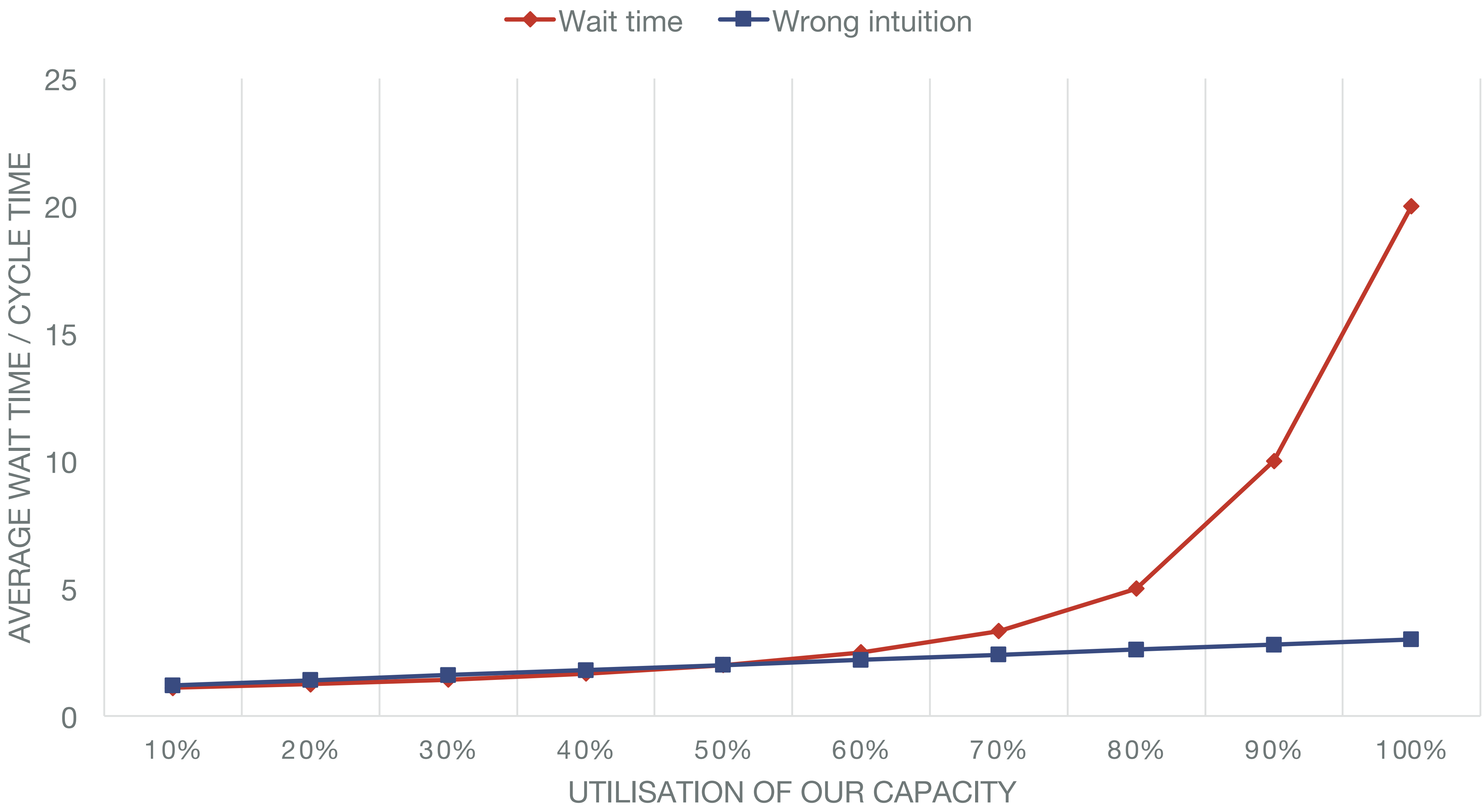
and

Kingman's Formula

Kingman’s Formula

$$Average\,Wait\,Time \propto \left(\frac{Utilisation}{1 - Utilisation}\right)$$

HIGH UTILISATION SHOOTS UP WAIT TIME



Over-utilisation causes jams

$$Average\,Wait\,Time \propto \left(\frac{Utilisation}{1 - Utilisation}\right)$$

Utilisation is hard to manage

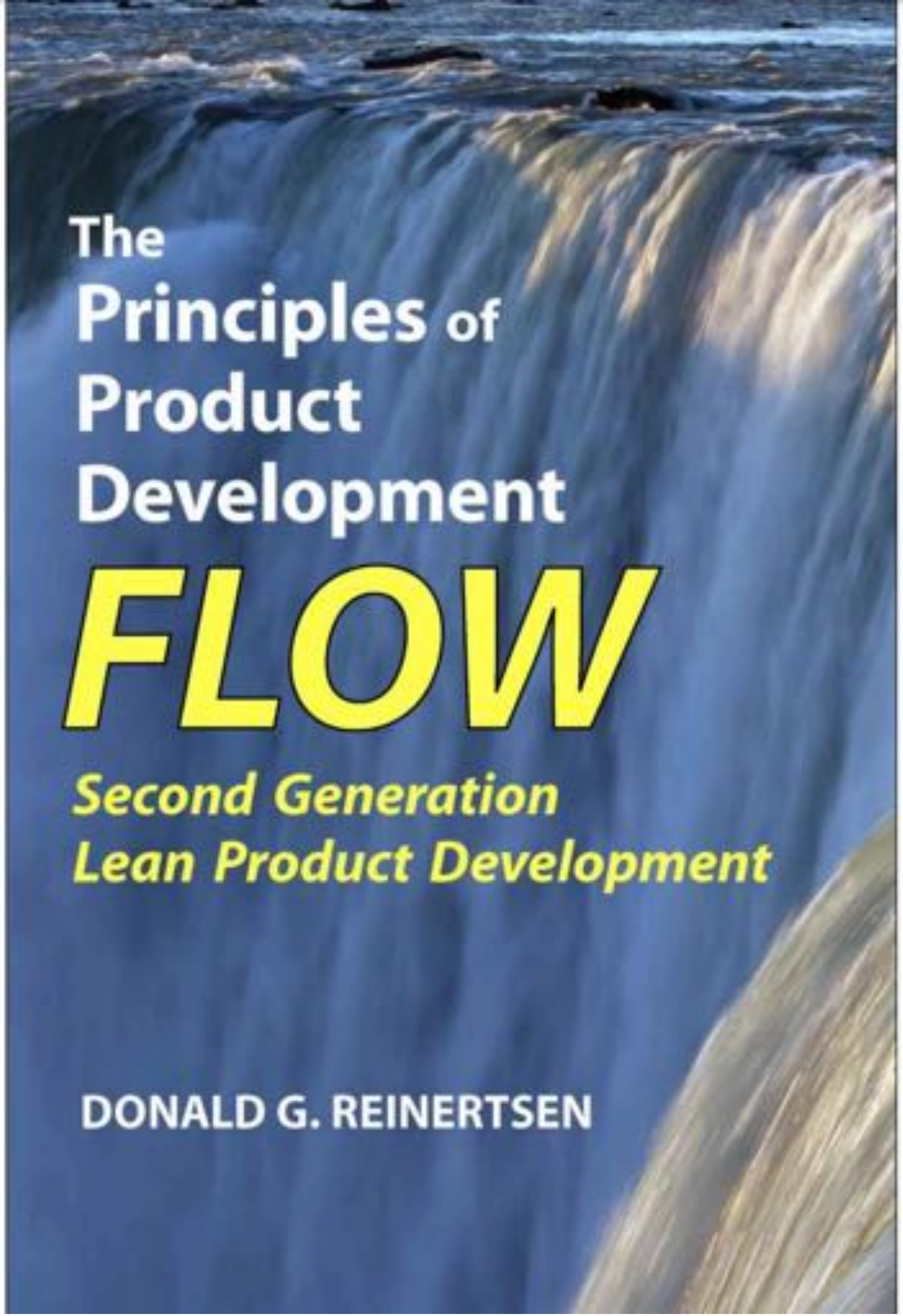
Little’s law

$$Avg\,Wait \approx \frac{Avg\,Work\ in\ Progress}{Avg\,Troughput}$$

Reduce *Work in Progress*
to
reduce *Utilisation*
to
reduce *Wait Time*

“Queues are the root cause of the majority of economic waste in product development.”

Donald G. Reinertsen



The Principles of Product Development

FLOW

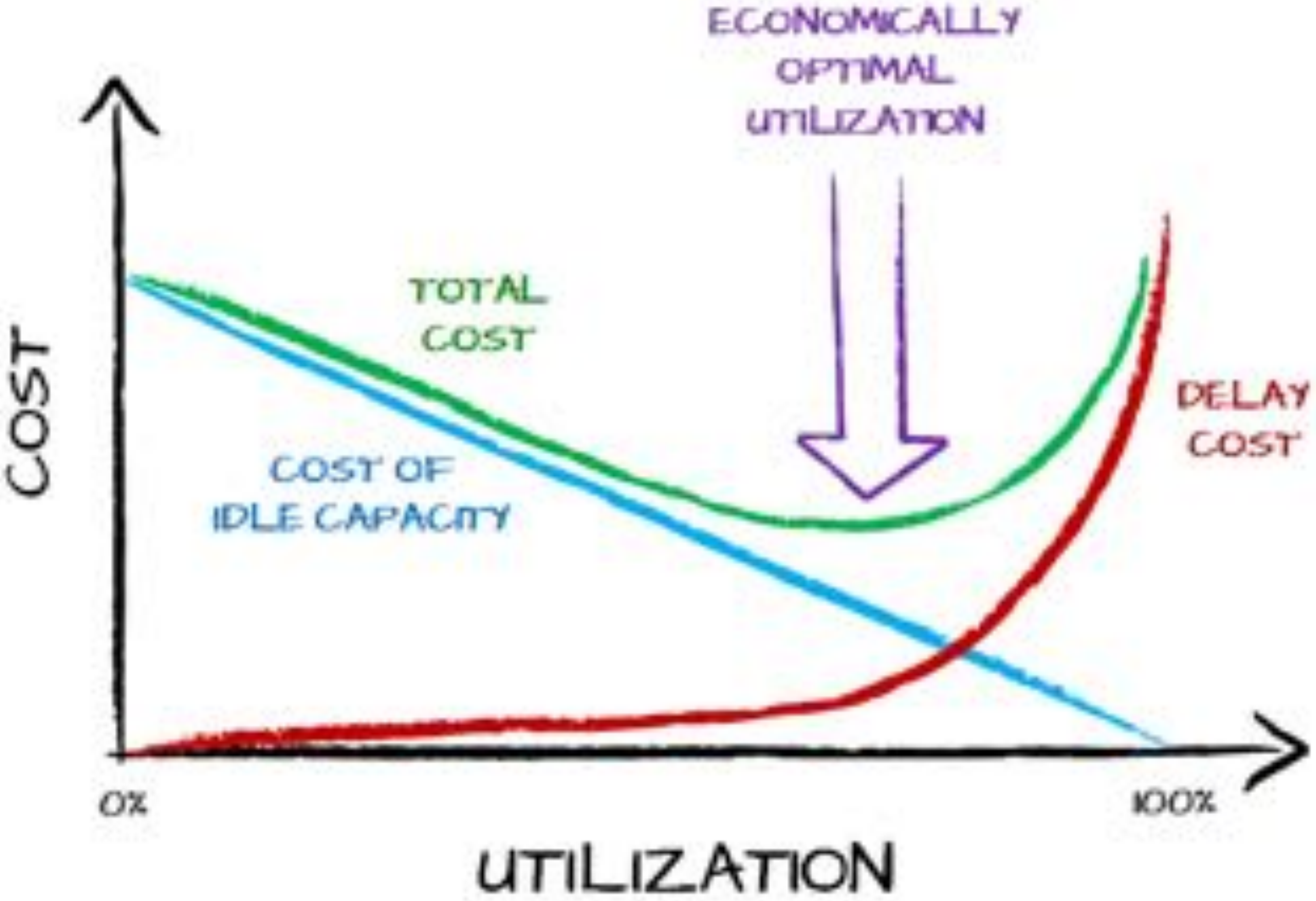
Second Generation Lean Product Development

DONALD G. REINERTSEN

In summary

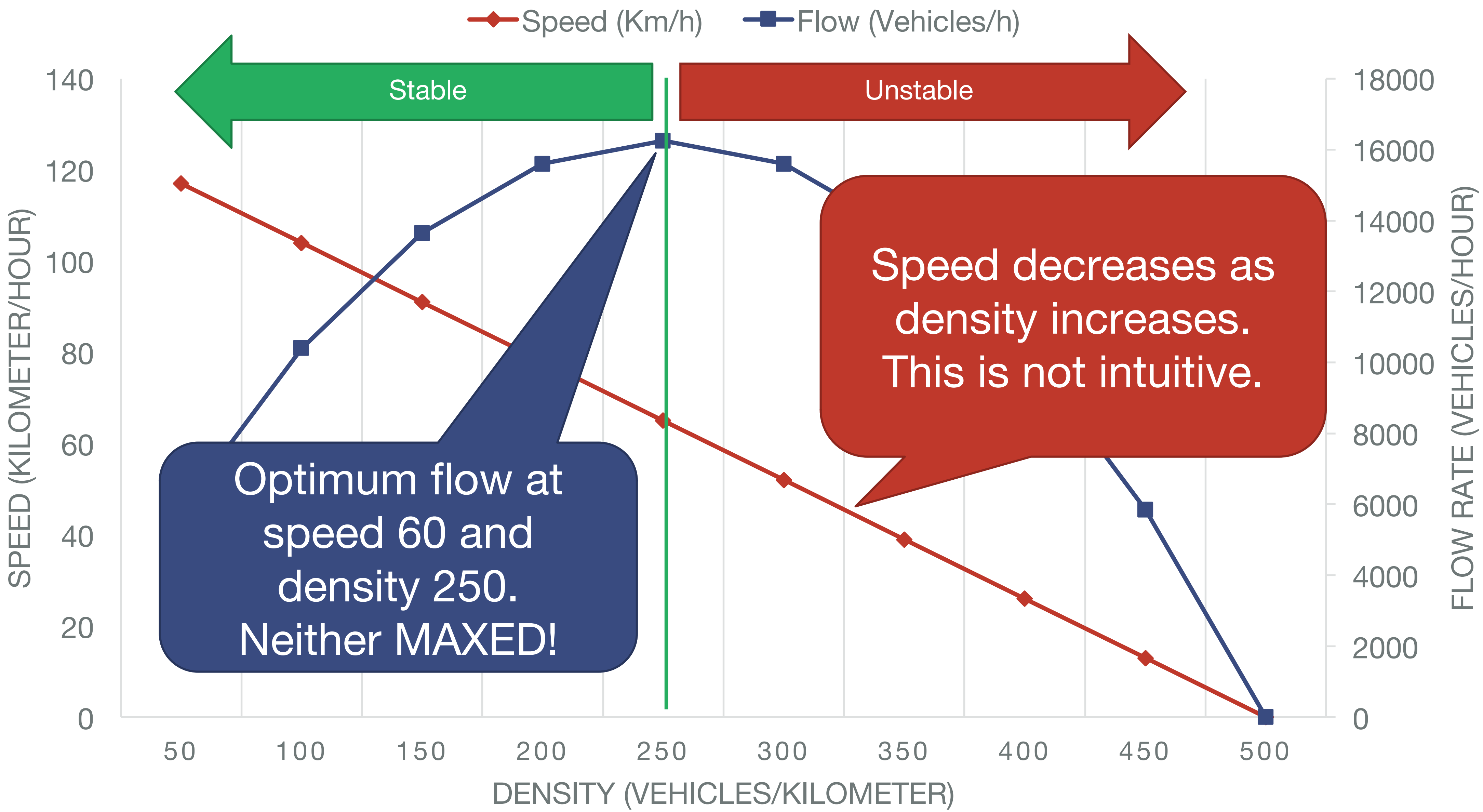
Avoid congestion and increase flow

- by reducing work in progress and
- reducing utilisation


ECONOMICALLY
OPTIMAL
UTILIZATION
TOTAL
COST
DELAY
COST
COST OF
IDLE CAPACITY
COST
0%
100%
UTILIZATION

$$\frac{Vehicles}{Hour} = \frac{Meters}{Hour} \cdot \frac{Vehicles}{Meter}$$

OPTIMAL FLOW



Optimum flow at speed 60 and density 250. Neither MAXED!

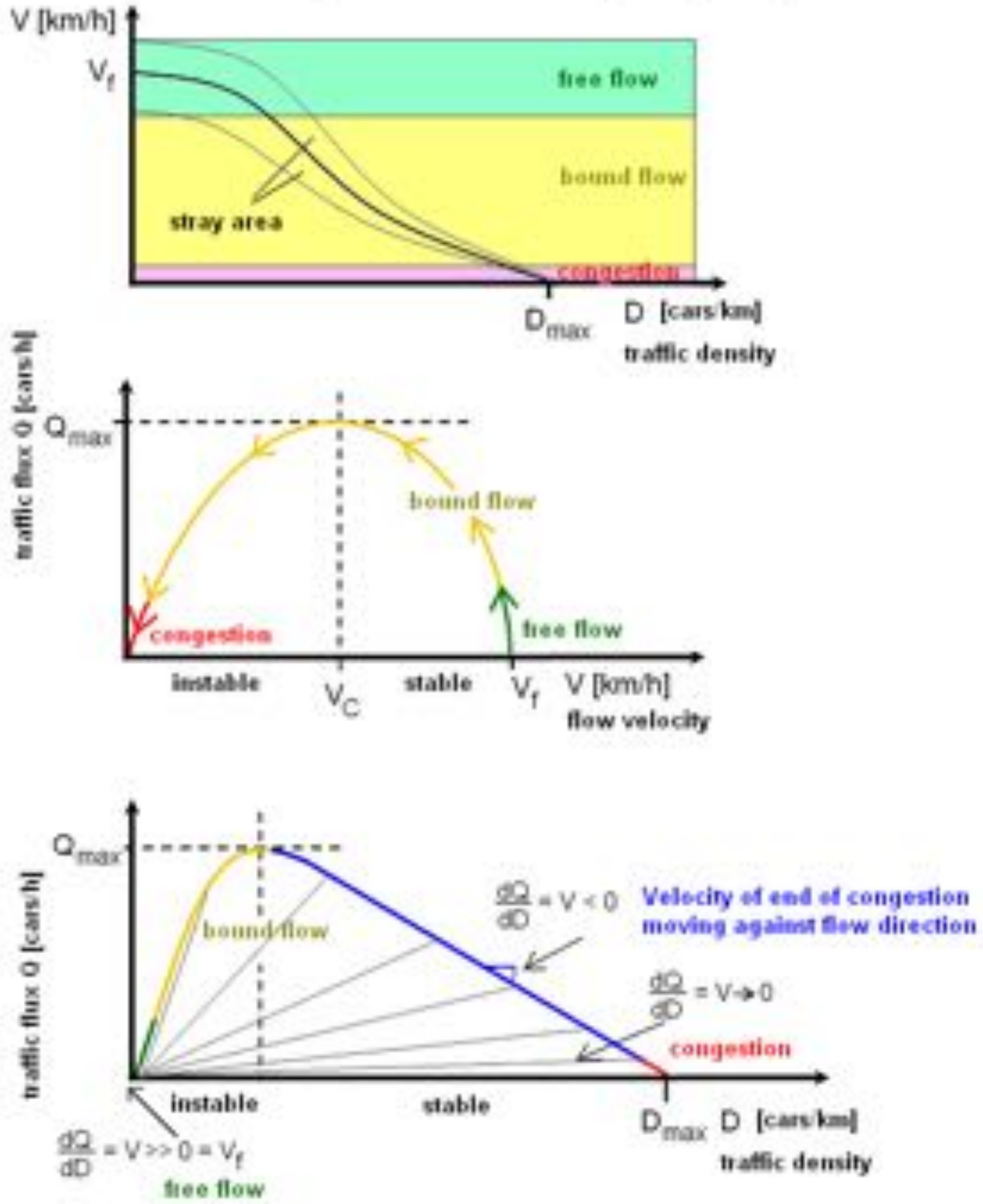
Speed decreases as density increases. This is not intuitive.

$$\mathit{Speed} = (1 - \mathit{Density} \;.\; \mathit{JamDens}) \;.\; \mathit{MaxSpeed}$$

Fundamental diagram of traffic flow

Fundamental equation of traffic flow: $\boxed{Q = D \cdot V}$

Source: Hendrik Ammoser, Fakultät Verkehrswissenschaften, Dresden, Germany



V_f = "free velocity" - maximum velocity on free lane, selectable by the driver depending on car, skill etc.

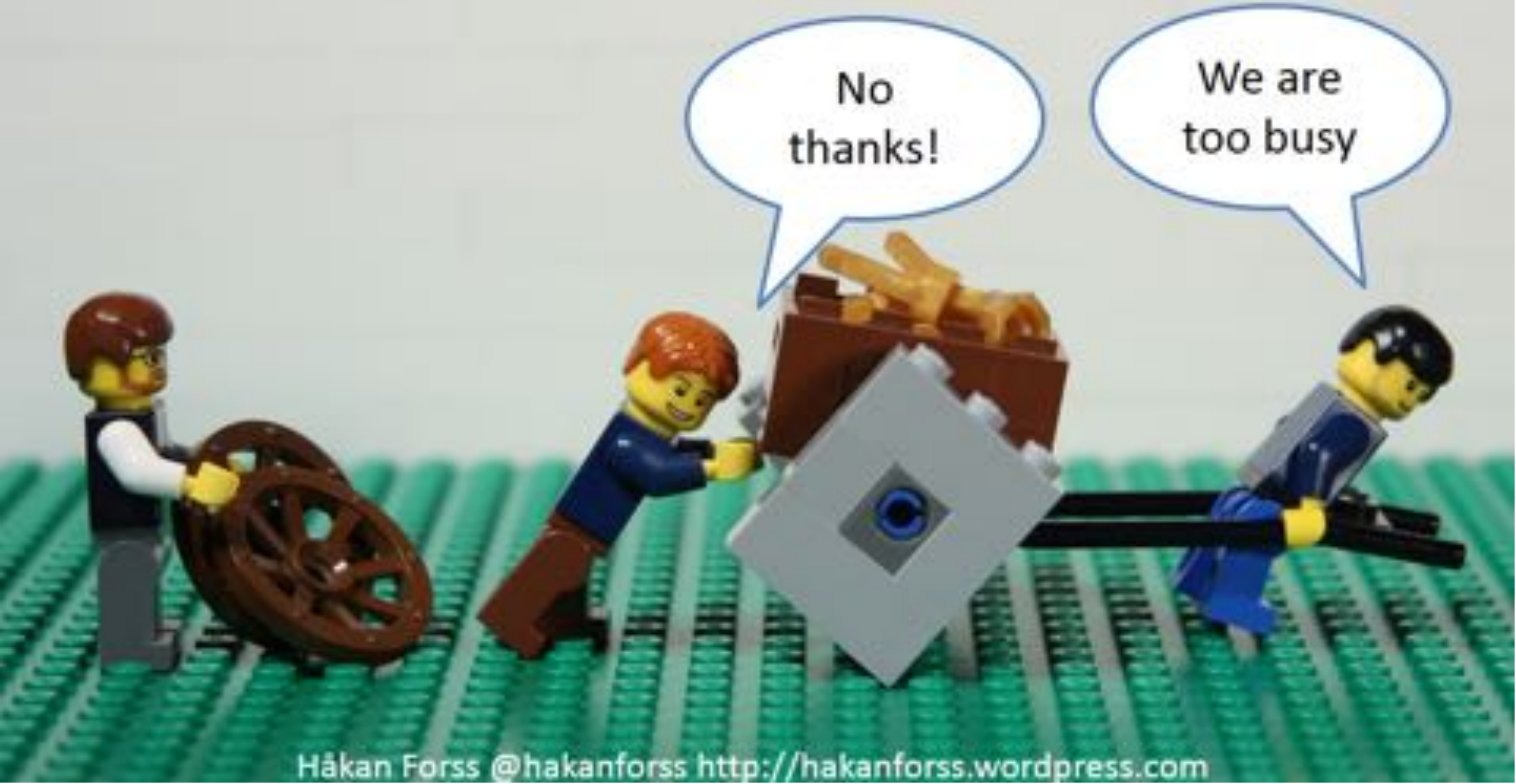
V_C = "critical velocity" with maximum traffic flux (about 70...100 km/h)

How does this apply to software?

- $Flow = Speed. Density$
- $Velocity = Throughput = Flow$ $(avg\ \#\ features\ delivered\ per\ week)$
- $WIP = Density$ $(avg\ \#\ features\ started\ but\ not\ completed)$
- $Cycle\ time\ = \frac{1}{Speed}$ $(avg\ time\ it\ takes\ to\ complete\ a\ feature)$
- $Thoughput = \frac{WIP}{Cycle\ Time}$ $(aka\ Little's\ Law)$

$$\boldsymbol{Velocity = \frac{Avg\ features\ in\ progress}{Avg\ time\ to\ complete\ a\ feature}}$$

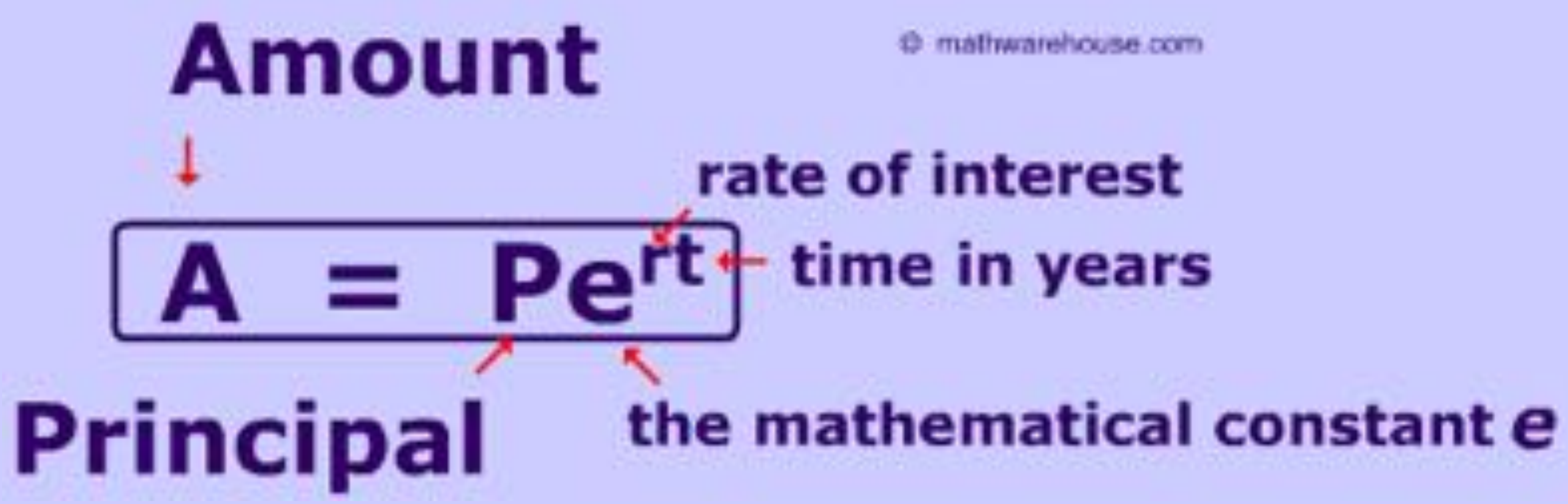
Are you too busy to improve?

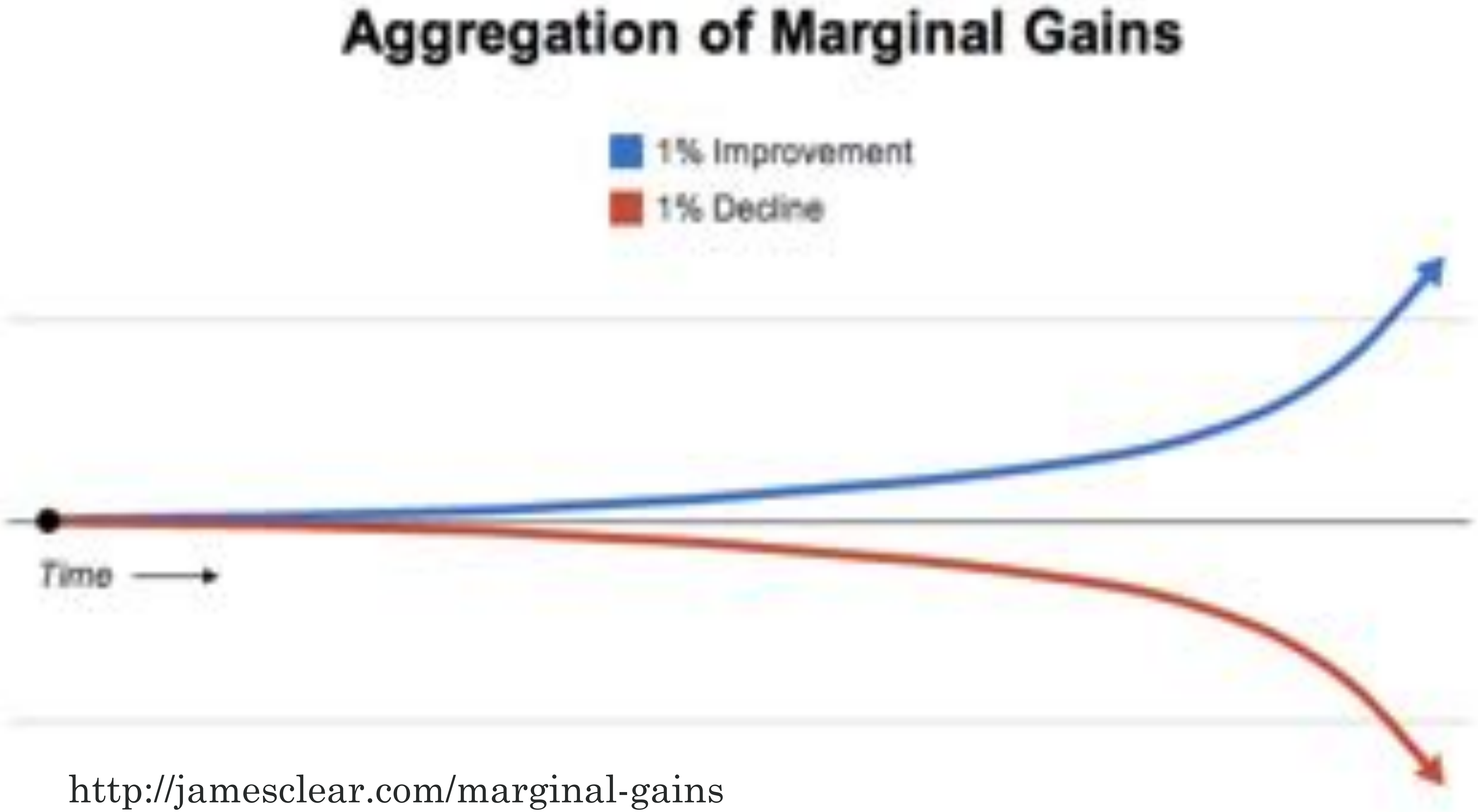


Improve continuously with

$$A = Pe^{rt}$$

Continuous compound interest





http://jamesclear.com/marginal-gains

$$Improvement = e^{rt}$$

- r = 1% per week (improvement rate)
- t = 104 weeks/2 years (time in weeks)

This gives us

- 294% improvement over 2 years!

$$e^{rt}$$

- r = -1% per week (decay rate)
- t = 104 weeks/2 years (time in weeks)

This gives us

- 34% of original
- Degraded 66% over 2 years!

Is this how technical debt behaves?

In summary

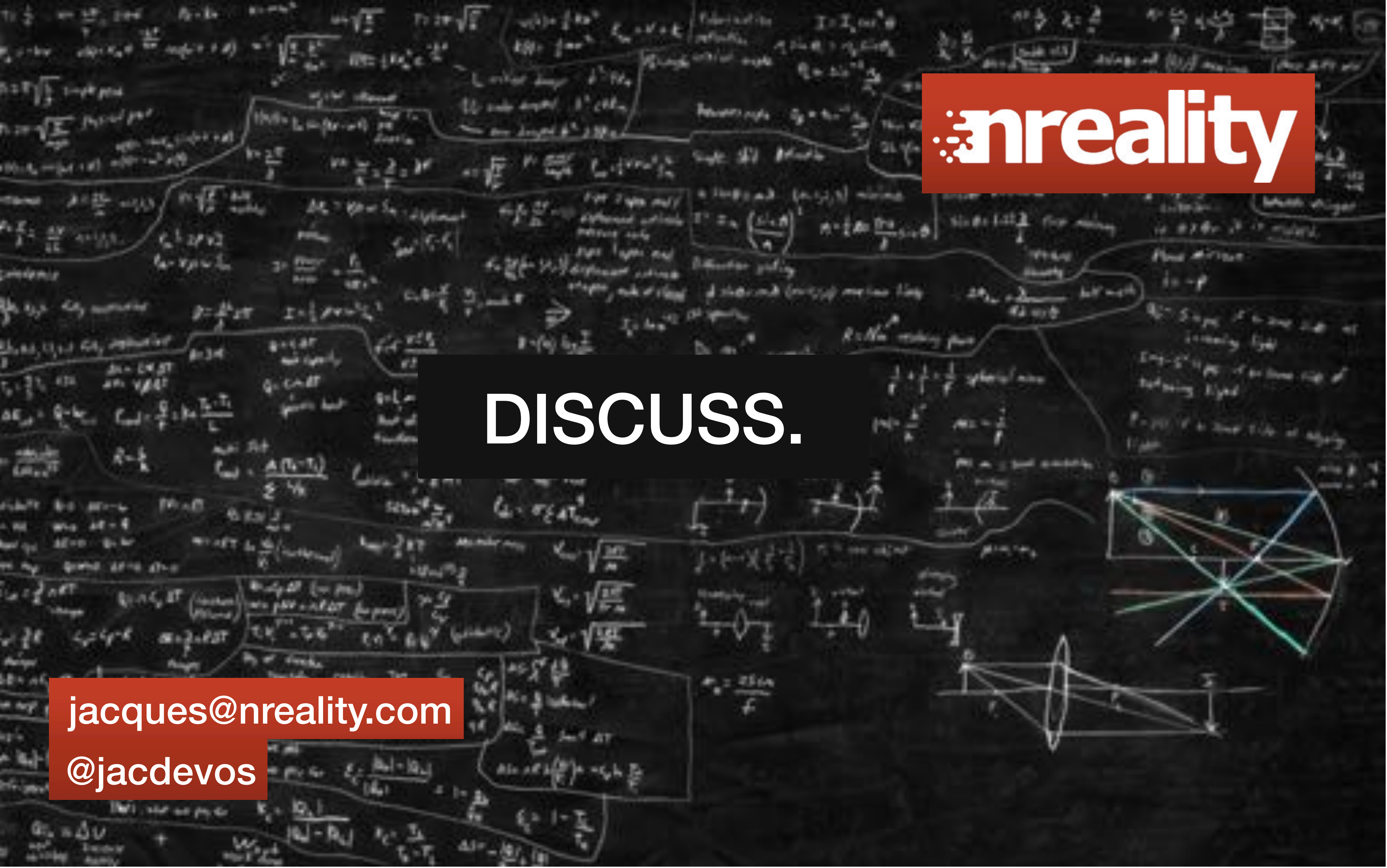
Improve 1% every week.

Improvements compound.

- Simplify by reducing dependencies
- You need quality to keep going fast
- Do less at once to go fast
- Continuous improvement compounds

Other equations

- Entropy
- Bayesian Inference
- The software engineering equation
- Yours?



nreality

DISCUSS.

jacques@nreality.com

@jacdevos