

# Flow & Lean software development

Thomas Luvö - @tomluvoe - [tom@samoh.se](mailto:tom@samoh.se)

# Recap

- The theme of my lectures has been to argue that traditional software project methods are not good enough to deliver high quality software products
- To show that in steps
  1. Scrum
  2. Traditional project management methods
  3. Agile software development methods
  4. Lean software development methods (today)

# Software development realities

- Product requirements will change between specification of a product and delivery
- Humphrey's Requirements Uncertainty Principle: A system can not fully be understood before it has been used
- Ziv's Uncertainty Principle for Software Development: That uncertainty and unpredictability is always part of software development
- Wegner's Lemma: It is not possible to fully specify an interactive system

# Traditional projects

- Traditional project management methods (waterfall etc.) all have stages in common
  1. Plan
  2. Implement, test and documentation
  3. Deploy and maintenance

# Assumptions in traditional project management methods

1. Requirements can be defined up front
2. That requirements do not change during the project
3. The customers know what they want before they see it

# Scrum (Agile)

- Scrum is an iterative and incremental software project framework for self-organizing and cross-functional teams
- Scrum rests on three principles
  1. Transparency
  2. Inspection
  3. Adaptation

# Agile software development

- Agile does not require that all requirements are known from the start
- Agile does support changing requirements
- Agile support that customers may change their mind during the project

# Overview

	Agile	Plan driven	Formal
Criticality	Low	High	Extreme
Experience	High	Low	High
Requirements	Dynamic	Static	Few
Size	Small	Large	
Culture	Adapt	Order	Quality

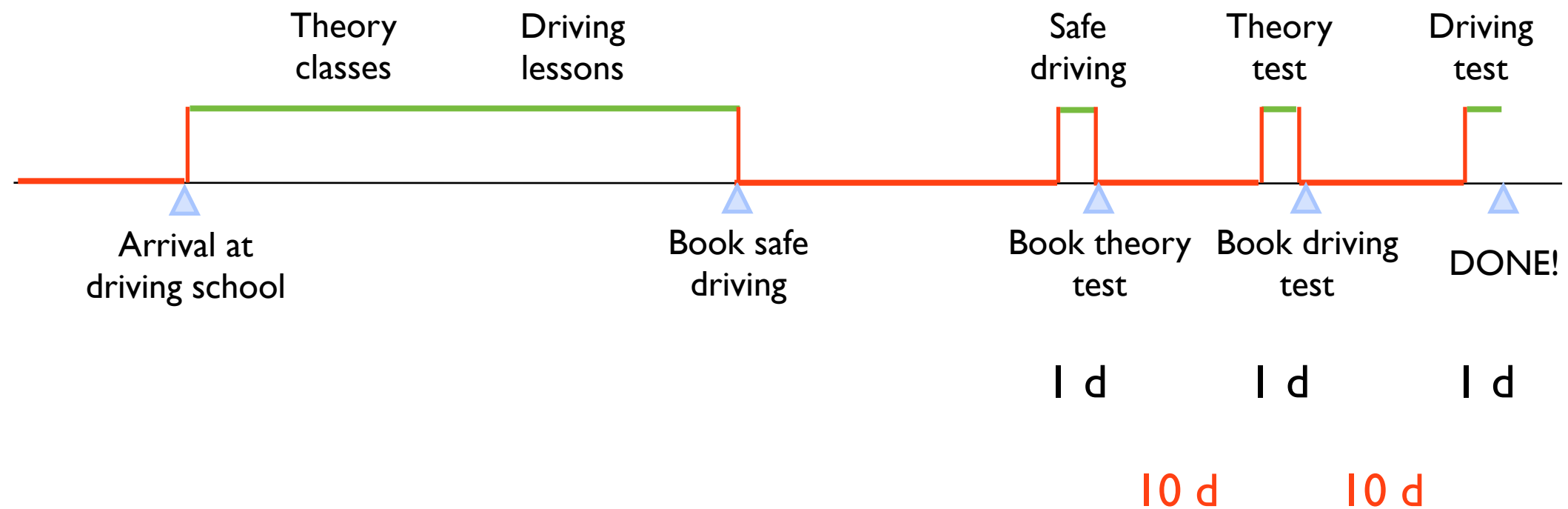


# Today

- Flow
- Lean software development

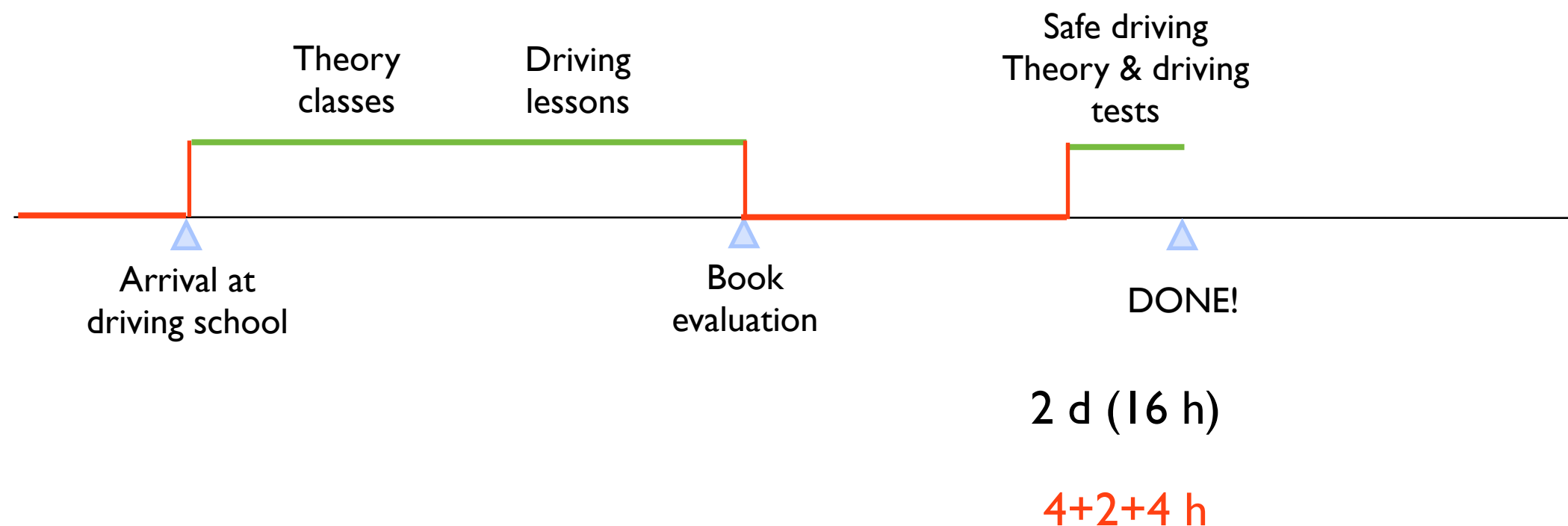
# What is flow?

# Driver license (1998)



Total time = 23 days  
Waiting = 20 days  
Efficiency = 0.09

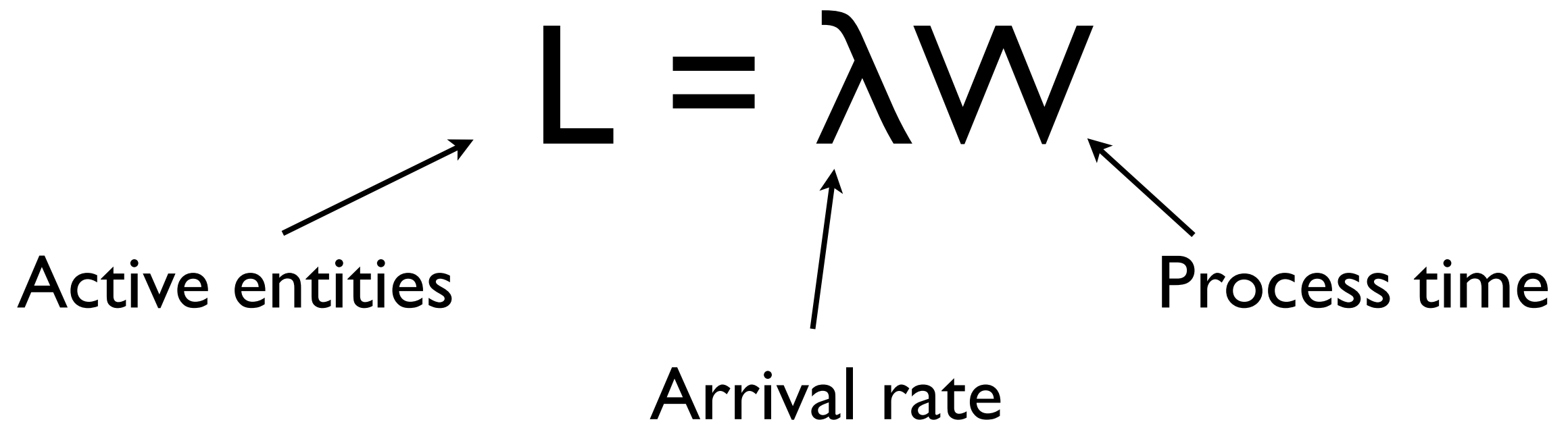
# Driver license (1998)



Total time = 16 h  
Waiting = 6 h  
Efficiency = 0.62  
21 days faster!

# What affects flow?

# Little's law



The diagram shows the equation  $L = \lambda W$  in a large, bold, black font. Three arrows point from descriptive labels to the variables in the equation: an arrow from 'Active entities' points to the 'L', an arrow from 'Arrival rate' points to the ' $\lambda$ ', and an arrow from 'Process time' points to the 'W'.

$$L = \lambda W$$

Active entities

Arrival rate

Process time

# Little's law

$$W = L / \lambda$$

The diagram illustrates Little's Law with the equation  $W = L / \lambda$ . Three labels are positioned below the equation, each with an arrow pointing to a specific part of the formula: 'Cycle time' points to the variable  $W$ , 'Active items' points to the variable  $L$ , and 'Arrival rate' points to the variable  $\lambda$ .

Cycle time

Active items

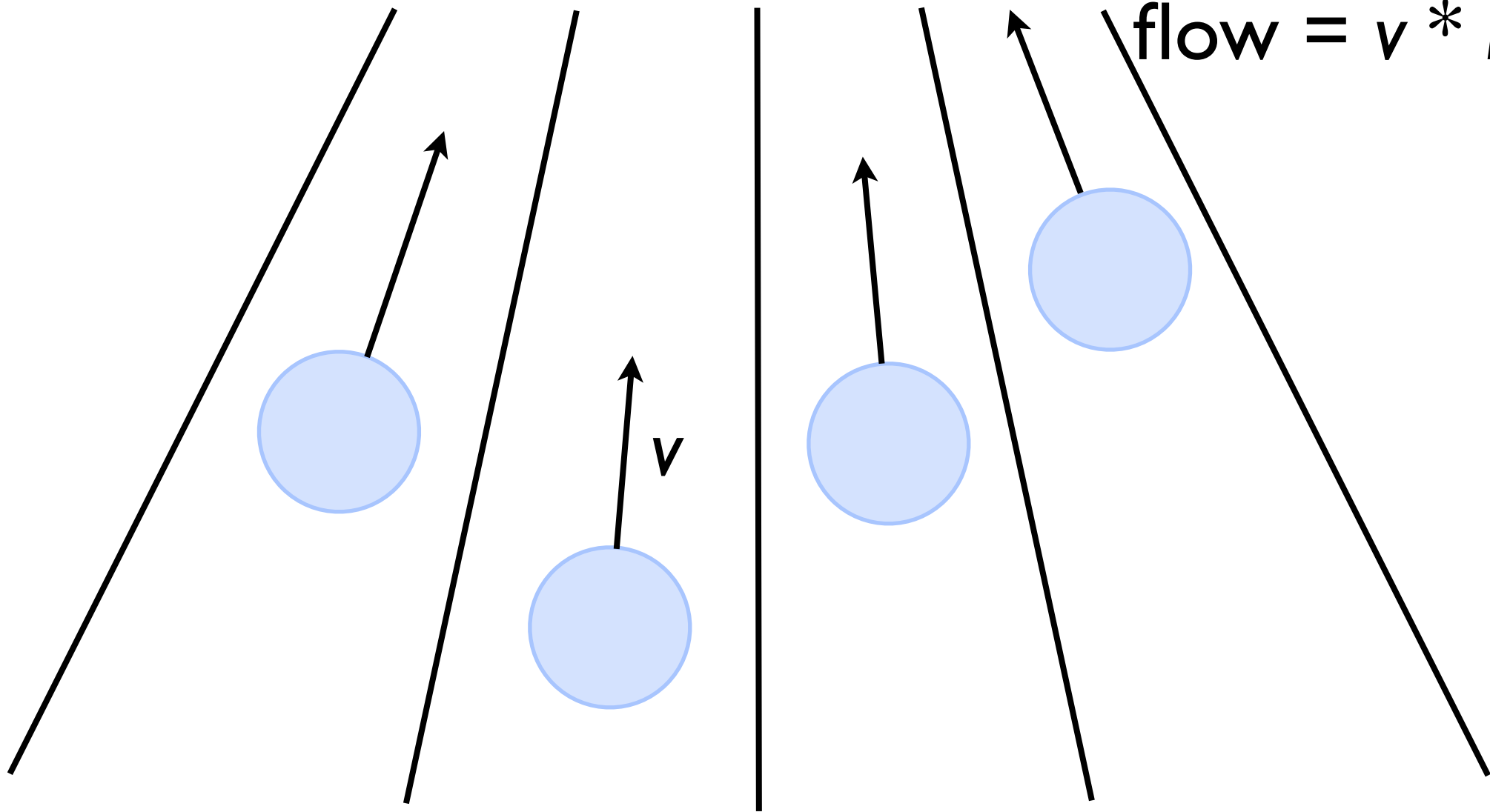
Arrival rate

# Flow on a highway

$$i = 4$$

$$v = 100$$

$$\text{flow} = v * i = 400$$

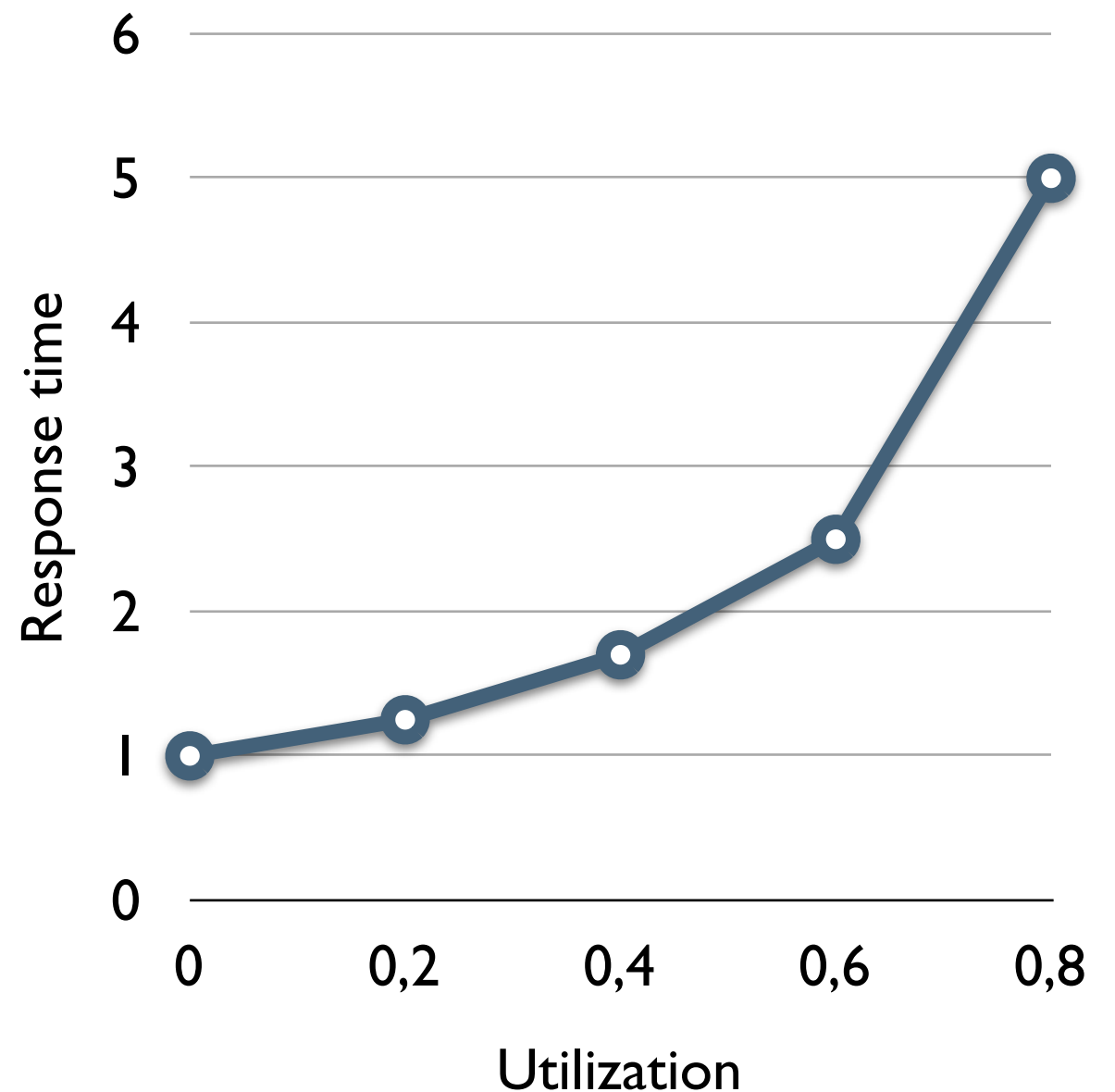




# M/M/1 queue

- $\lambda$  = arrival rate
- $\mu$  = service rate (1/W)
- $\rho = \lambda / \mu$  = utilization (Little's law)
- Response time =  $1 / (\mu - \lambda)$

Variation?

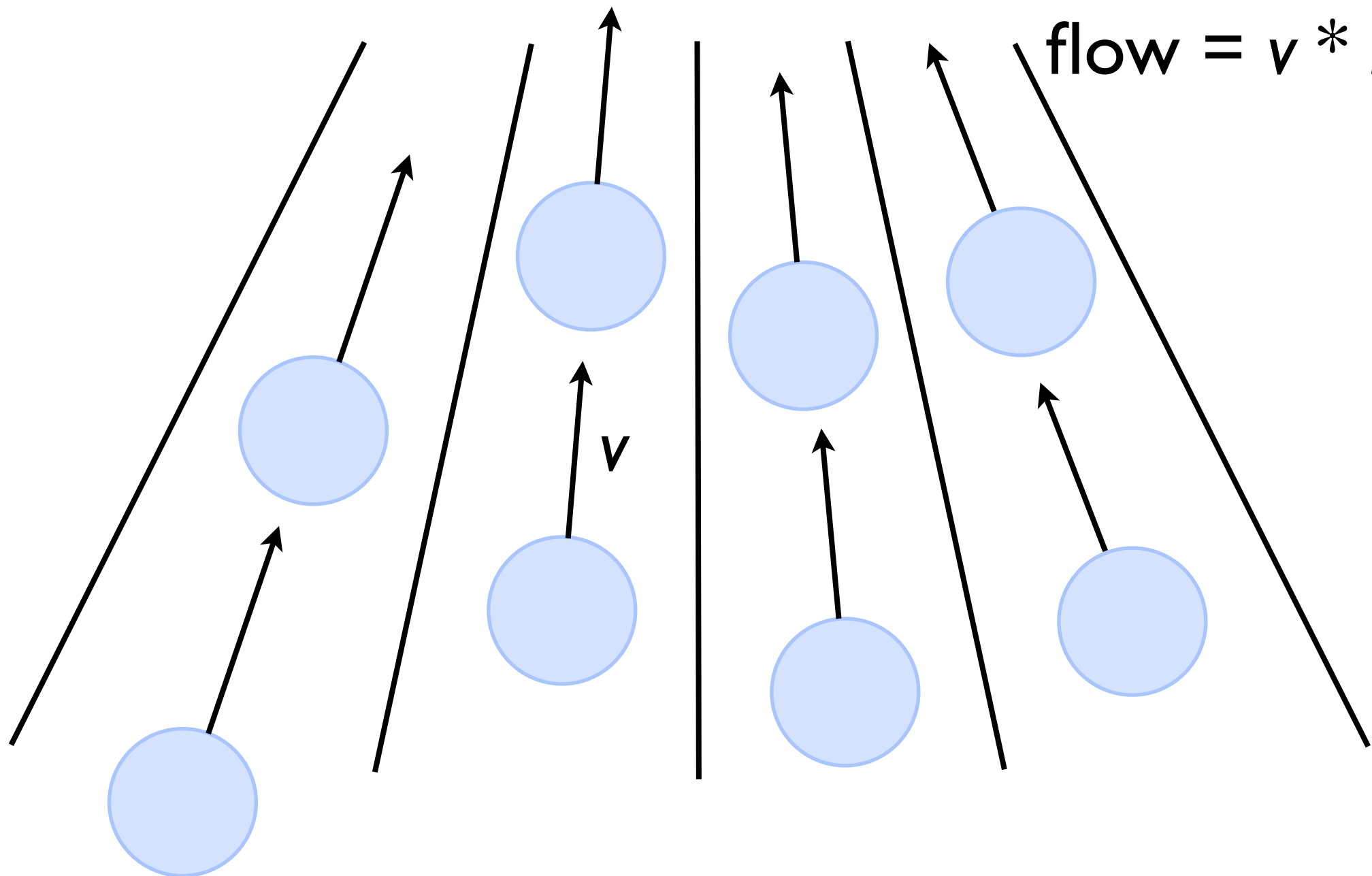


# Flow on a highway

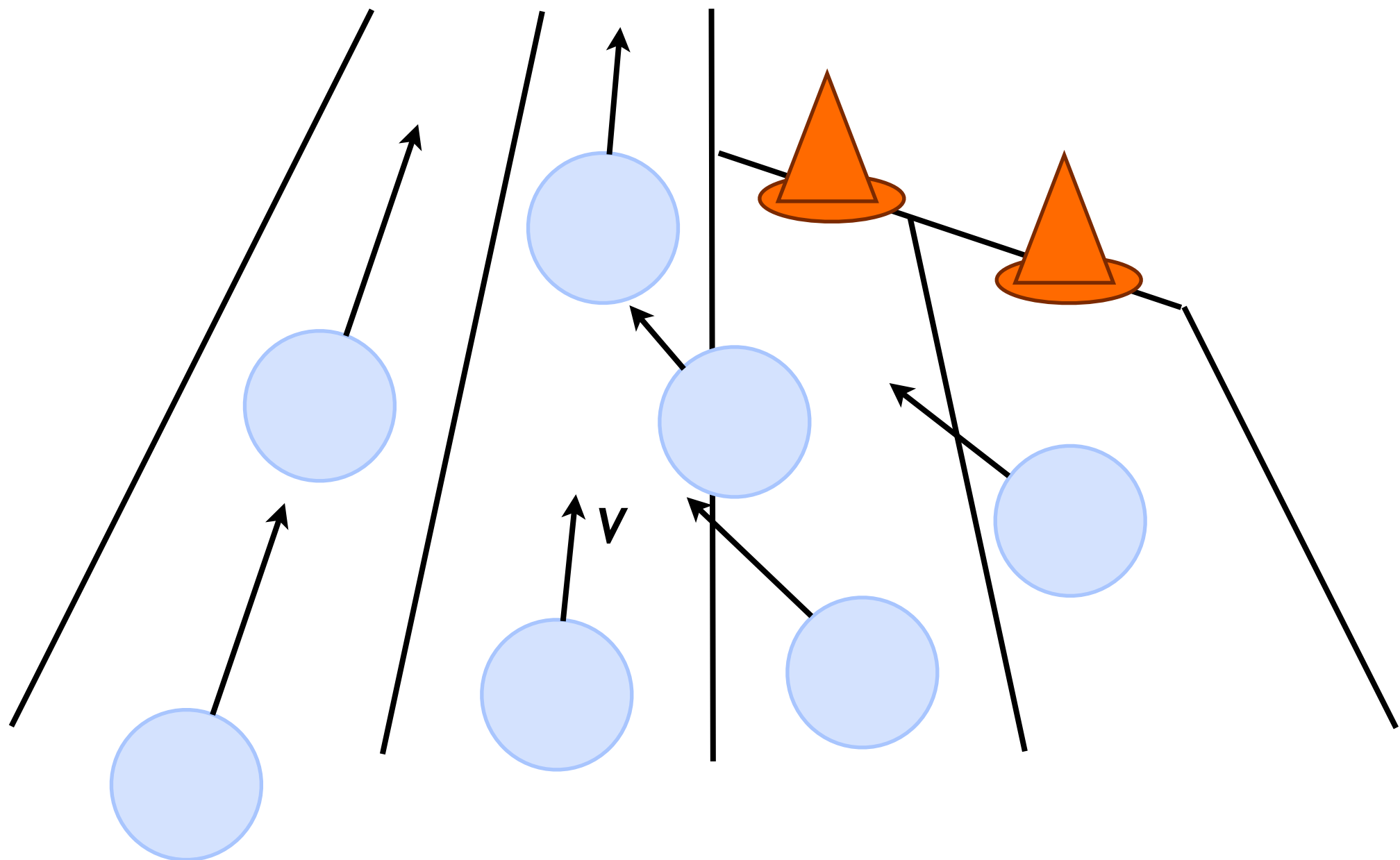
$$i = 8$$

$$v = 50?$$

$$\text{flow} = v * i = 400?$$



# Bottlenecks (queues)



# Flow summary

- Flow is good for development, the higher the flow, the quicker the customer will get features and the development company paid
- Flow is affected negatively by
  - Queues
  - Process time (item size)
  - High utilization
  - Variation in item size
- How does this relate to Scrum
  - Cross-functional teams, break down user stories, teams pull items in Sprint Planning to have the right amount of work, user stories are approximately the same size

# Lean software development

# Lean

- Toyota
  - 331876 employees
  - 13th largest company in the world
  - Largest automobile manufacturer (2012)
  - Manufactures cars since 1933
- Recognized as an industry leader in production.

# Lean software development

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

# Eliminate waste

- Value: Actions, thoughts etc. that create the product the customer is willing to pay for
- Waste: Everything else
- Examples of waste
  - Waiting
  - Handoffs, hand-overs
  - Unused features
  - Multitasking
  - Work in progress



# Amplify learning

- Improve the software development environment through learning
- Learn faster by using short iterations, both internal test and customer feedback
- Instead of adding more documentation or plans, try different ideas through experimentation

# Decide as late as possible

- Decisions should be made on facts and not on guesses
- When working in a complex environment (such a software development) we face many uncertainties
- Therefore decisions should be delayed as much as possible

# Deliver as fast as possible

- Quality vs. speed?
- Speed
  - Late decisions
  - Fast feedback
  - Learn faster
  - Customer can make up their minds faster
  - Minimize waste

# Empower the team

- The people on the front line knows the details and are best equipped to make the correct decisions
- Decisions are therefore allowed to be made as late as possible

# Build integrity in

- Software that really works, fulfills the customer's wishes and has conceptual integrity
- Maintain usefulness over time
- Key practice is refactoring

# See the whole

- A product is a sum of all parts and their interaction
- Avoid sub-optimization
- Do not only maximize performance in each domain/component alone

# Lean software development and Scrum

- |                                |  |
|--------------------------------|--|
| 1. Eliminate waste             | 1. Sprint planning with pull, all work in backlog, one team - no handovers |
| 2. Amplify learning            | 2. Retrospectives, short sprints, customer delivery at end of sprint       |
| 3. Decide as late as possible  | 3. The self-organizing teams make decisions                                |
| 4. Deliver as fast as possible | 4. Work in short iterations delivery possibility after each iteration      |
| 5. Empower the team            | 5. Self-organizing teams   |
| 6. Build integrity in          | 6. Refactoring as practice   |
| 7. See the whole               | 7. Use feature teams and not component teams                               |

# Summary



# Q&A