26th annual INCOSE international symposium

Edinburgh, UK

July 18 - 21, 2016

Antonio Martini

Jan Bosch

ARCHITECTURAL TECHNICAL DEBT IN SYSTEM ENGINEERING

Software center

AUREA

Boschonian

CHALMERS

UNIVERSITAS GOTHOBURGENSIS 1891

Software Center

Mission: Improve the software engineering capability of the Nordic Software-Intensive industry with an order of magnitude

Theme: Fast, continuous deployment of customer value

Success: Academic excellence

Success: Industrial impact

Tetra Pak

SAAB TECHNOLOGIES

verisure
ALARMS WITH IQ

AXIS COMMUNICATIONS

VOLVO

VOLVO

CHALMERS

MALMÖ UNIVERSITY

JEPPESEN
A BOEING COMPANY

LINKÖPINGS UNIVERSITET

MÄLARDALEN UNIVERSITY
SWEDEN

ERICSSON

GRUNDFOS

A Software Center Project (1)

Current participants from industry

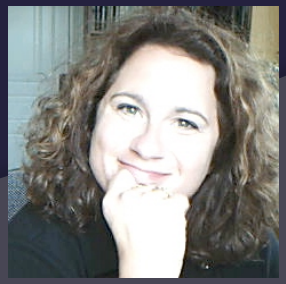
A Software Center Project (2)

Current research participants

- Antonio Martini
  - Postdoc at Chalmers
  - antonio.martini@chalmers.se

- Jan Bosch
  - Full Professor at Chalmers
  - jan@janbosch.com

- Teres Besker
  - PhD Candidate at Chalmers
  - besker@chalmers.se



G . U .

CHALMERS

Agenda

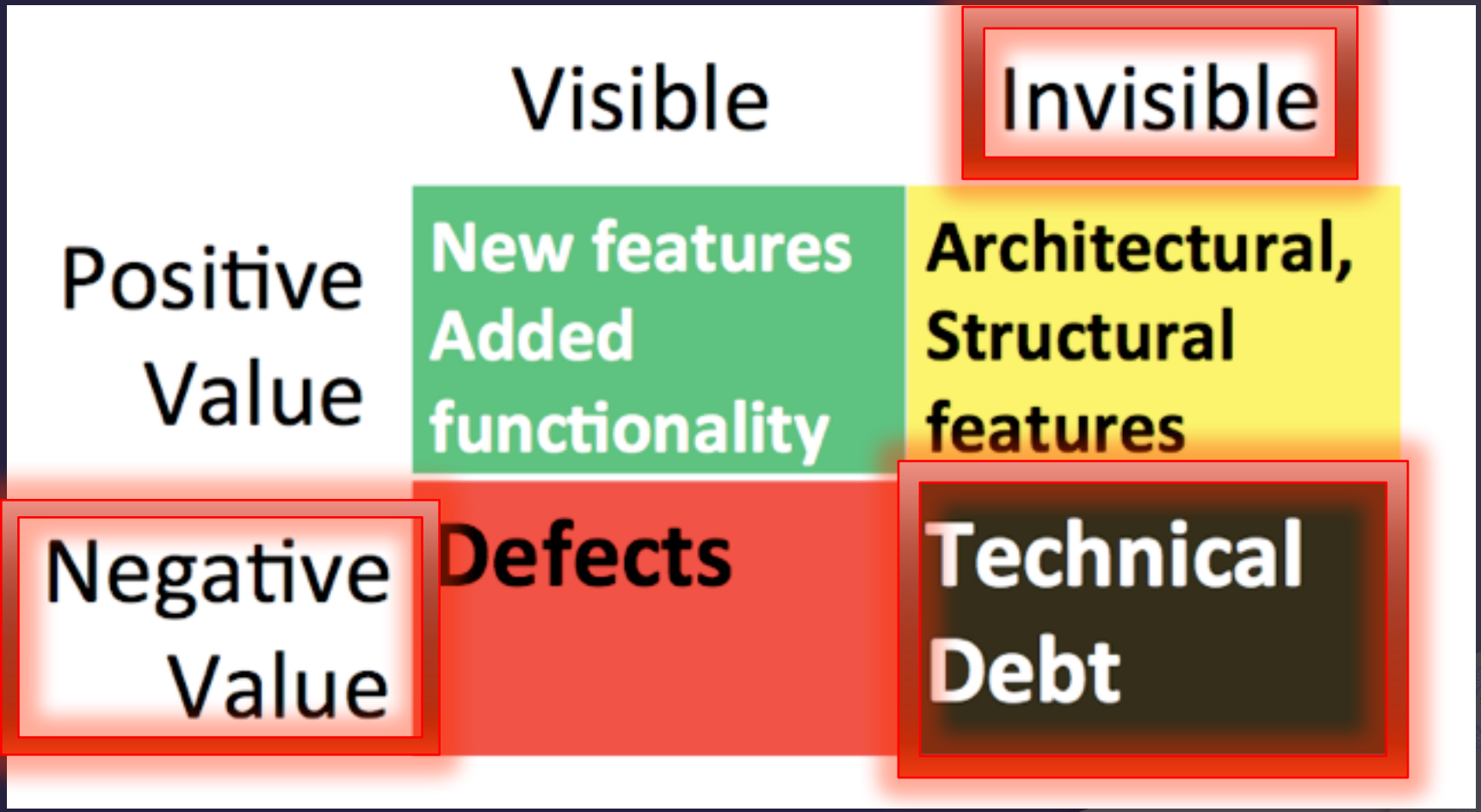
- Architectural Technical Debt (**ATD**)
  - What ***is*** it?
  - What ***causes*** it?
  - What are its ***consequences***?
  - How to ***manage*** it?
    - (short summary)

What is Architecture Technical Debt

G . U .

CHALMERS

What is Technical Debt?



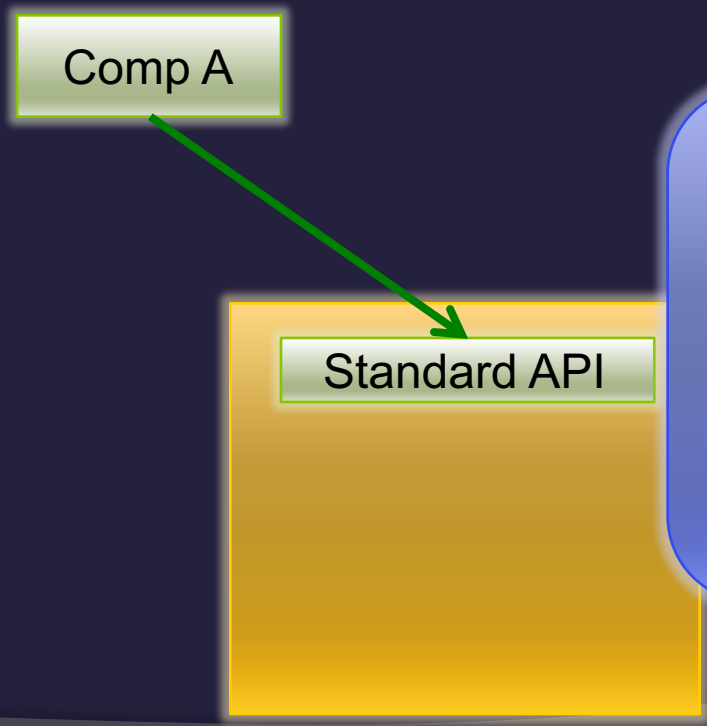
P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*

What is Architectural Technical Debt?

- Sub-optimal architectural solutions that
  - Have a beneficial impact on short-term goals but
    - aka “taking debt”
  - Have a negative impact in the medium-long run
    - aka “paying the interest”
- Better explanation: a (horror) story

Optimal architectural decision

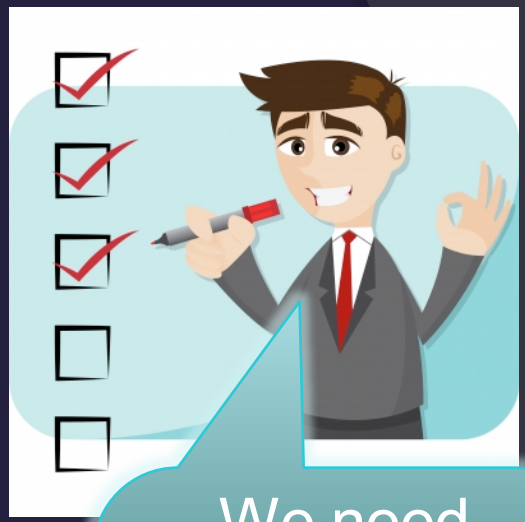
- Example:
  - Standard public API



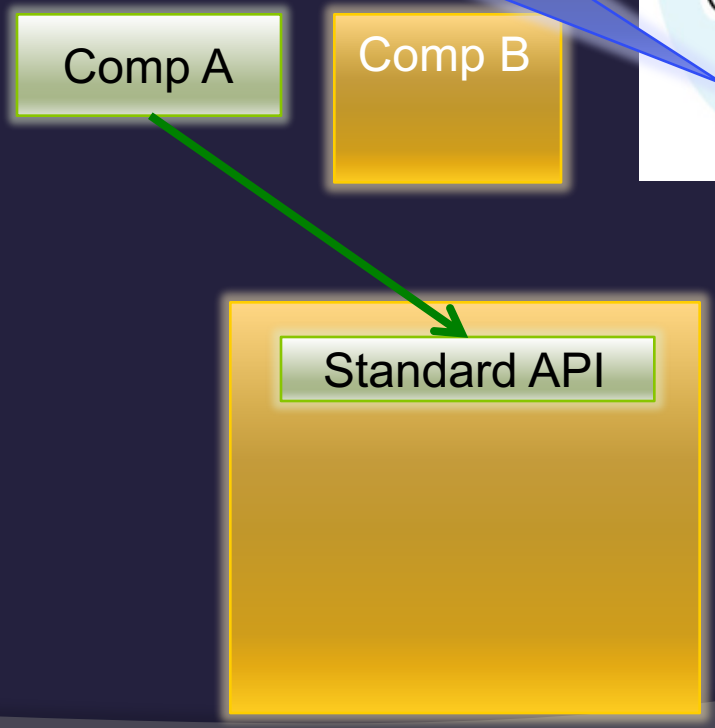
Let's put a standard API here… so later we can update the component independently

During feature development...

No problem, let's add a component B. The teams will use the standard API!

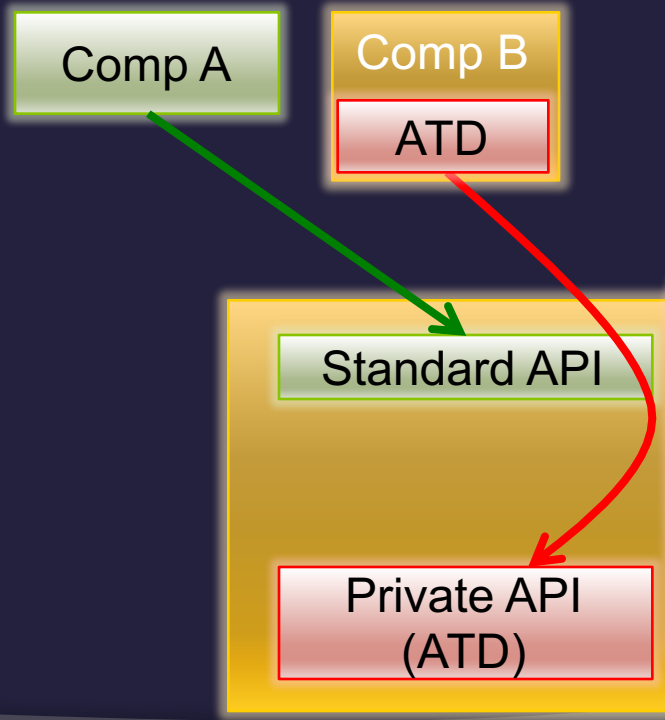


We need these new features! Our competitor is already delivering them!

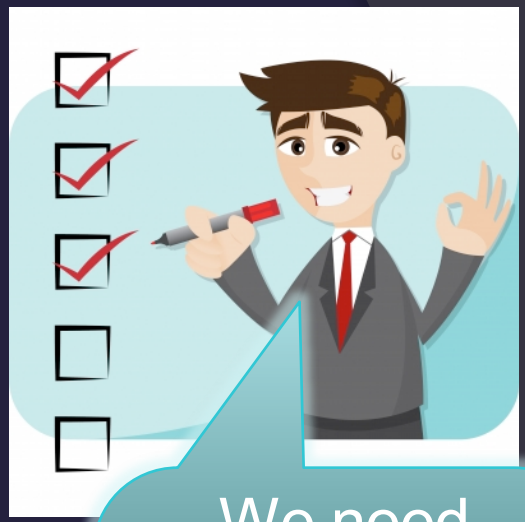


...with fast delivery comes...

- Deliver fast!



We have to deliver fast, let's use the private API… we'll change it later

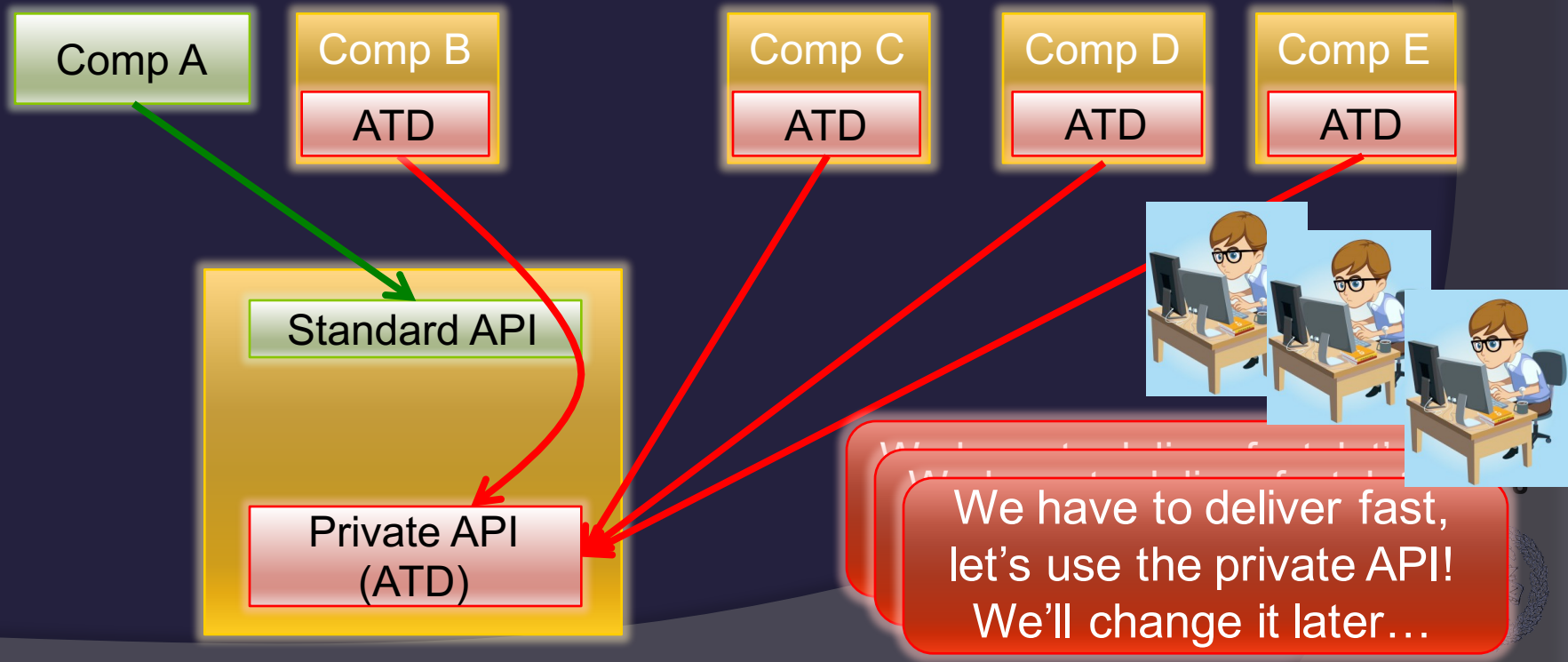


We need these new features! Our competitor is already delivering them!

Fast!

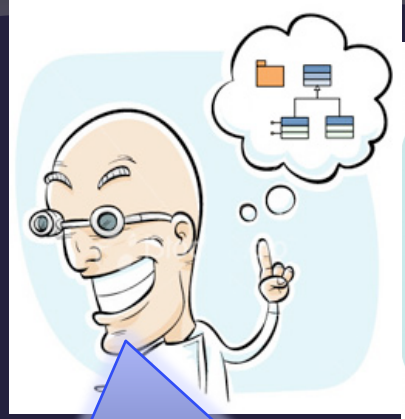
...the accumulation of sub-optimal decisions...

- The violation is spreading to many components



...until, one day...

- New requirement



Ok, we can replace this component. The teams used the standard API!

Comp A

Com

ATD

ATD

ATD

Com

AT

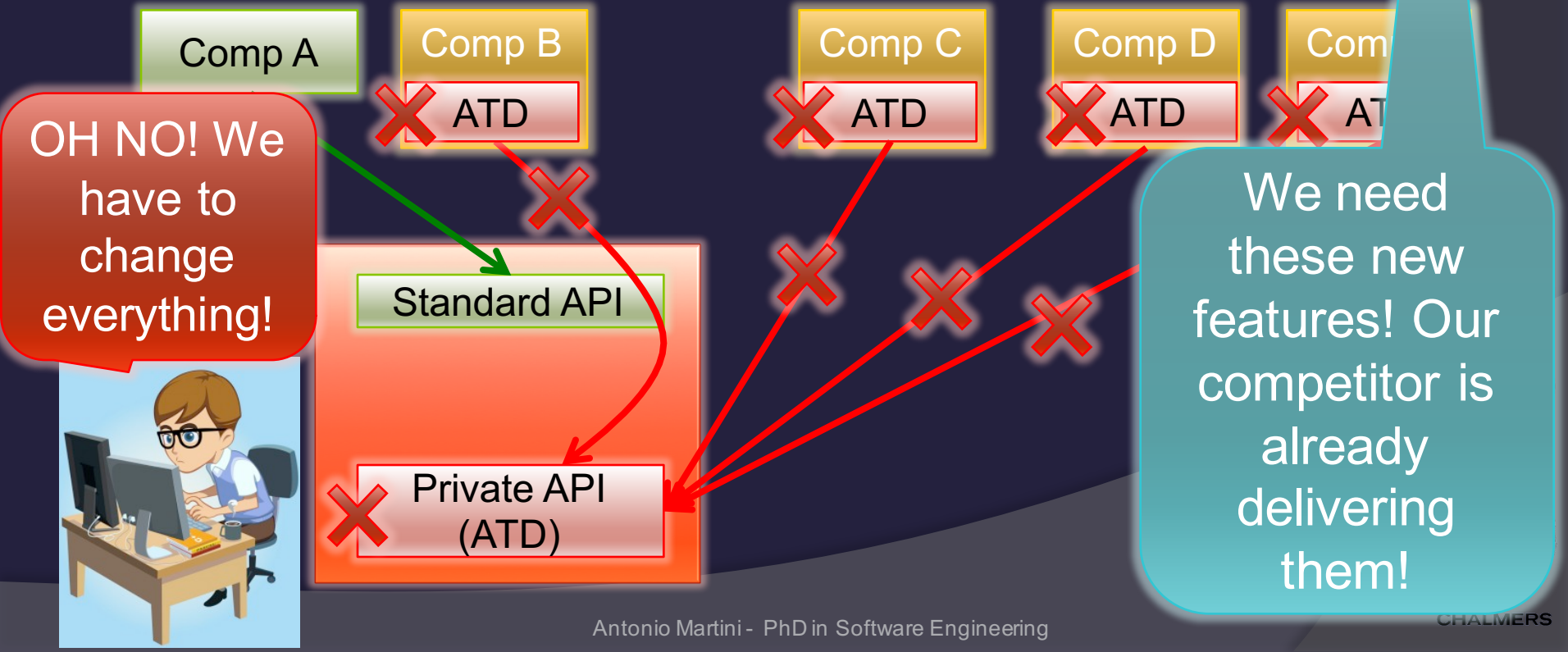
Standard API

Private API (ATD)

We need these new features! Our competitor is already delivering them!

...the development is not fast anymore...

- *Costly* to remove the violation and *difficult to estimate the impact*



...and a crisis starts.

We have to refactor, but we need time…

So should we refactor or continuing with other features?

Comp B

ATD

Comp C

ATD

Comp D

ATD

Comp

AT

Standard API

Private API (ATD)

Impossible to refactor now! We need to deliver the features!

Architecture Technical Debt in a real example

- Non-allowed dependencies = "Taking" the Debt
  - Save time by non-applying the optimal solution
- Cost of removing dependencies = Principal
  - How much does it cost to provide the optimal solution?
- Extra evolution cost
  - Replacing the component
- Other impacts = Interest
  - Increasing principal
  - Difficult estimation
  - Lead time increases

Architecture Technical Debt in a real example

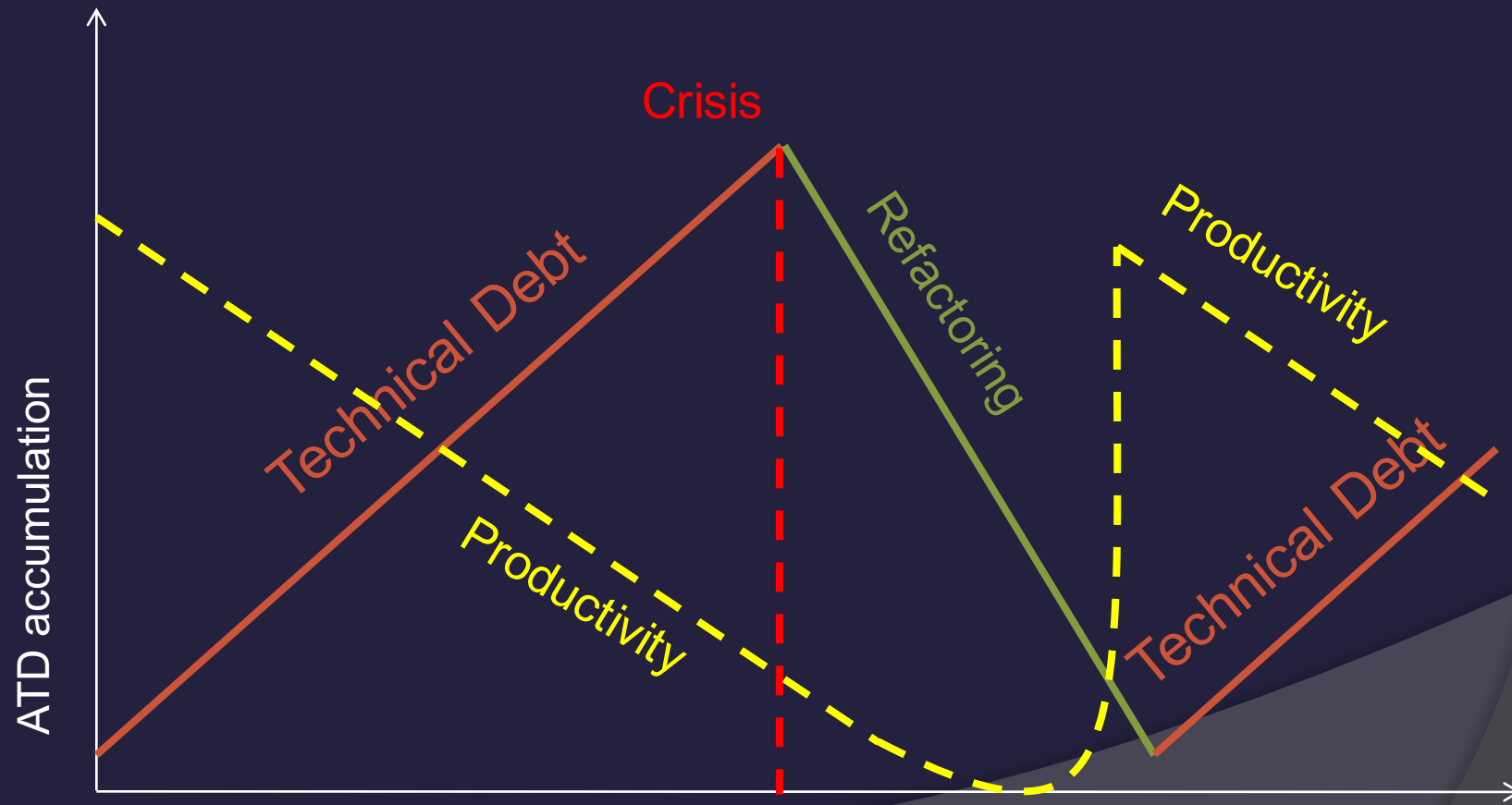
- Non-allowed dependencies = "Taking" the Debt
  - Save time by non-applying the optimal solution
- Cost of removing dependencies = Principal
  - How much does it cost to provide the optimal solution?
- Extra evolution cost
  - Replacing the component
- Other impacts
  - Increasing principal
  - Difficult estimation
  - Lead time increases

Interest-ing!

= Interest

When we have several teams and a big project...

- What happens then?



* Martini, A., Bosch, J., Chaudron, M., 2015. "*Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study*", Information and Software Technology Jounral

Causes of Architecture Technical Debt

G . U .

CHALMERS

Causes of accumulation?

- Business factors:
  - Uncertainty of use cases in the beginning
  - Business evolution
  - Time pressure: deadlines with penalties (urgency)
  - Priority of features over product architecture
  - Split of budget in Project and Maintenance
- Lack of specification/emphasis on critical architectural requirements
- Reuse of Legacy / third party / open source
- Parallel development
- Effects Uncertainty (unknown effects)
- Non-completed Refactoring
- External factors
  - Ex.: Technology evolution
- Human factor

* Martini, A., Bosch, J., Chaudron, M., 2015. "*Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study*", Information and Software Technology Jounral

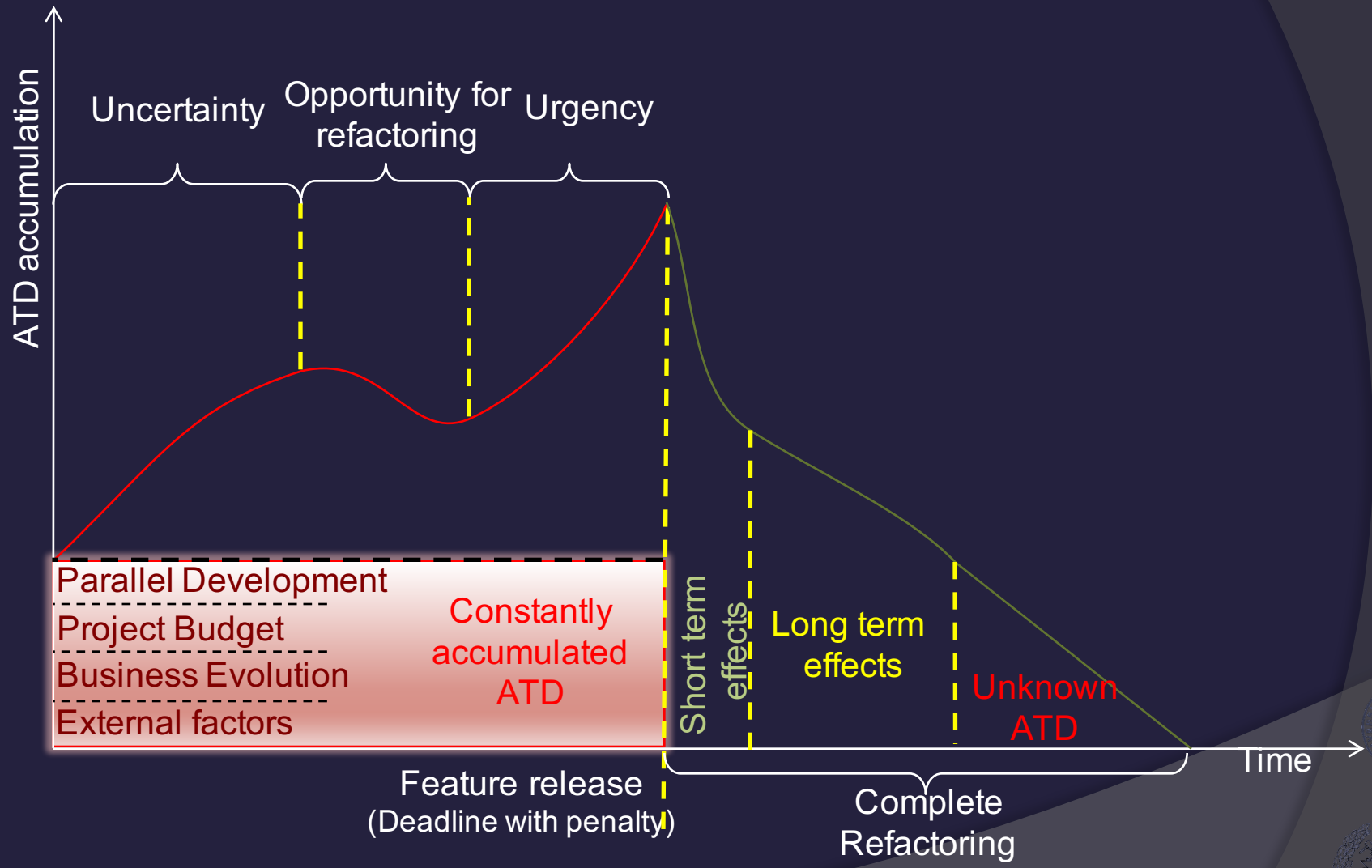
Causes of accumulation?

- Business factors:
  - Uncertainty of use cases in the beginning
  - Business evolution
  - Time pressure: deadlines with penalties (urgency)
  - Prio
  - Split
- Lack o
  requir
- Reuse
- Parallel development
- Effects Uncertainty (unknown effects)
- Non-completed Refactoring
- External factors
  - Ex.: Technology evolution
- Human factor

Technical Debt is inevitable!

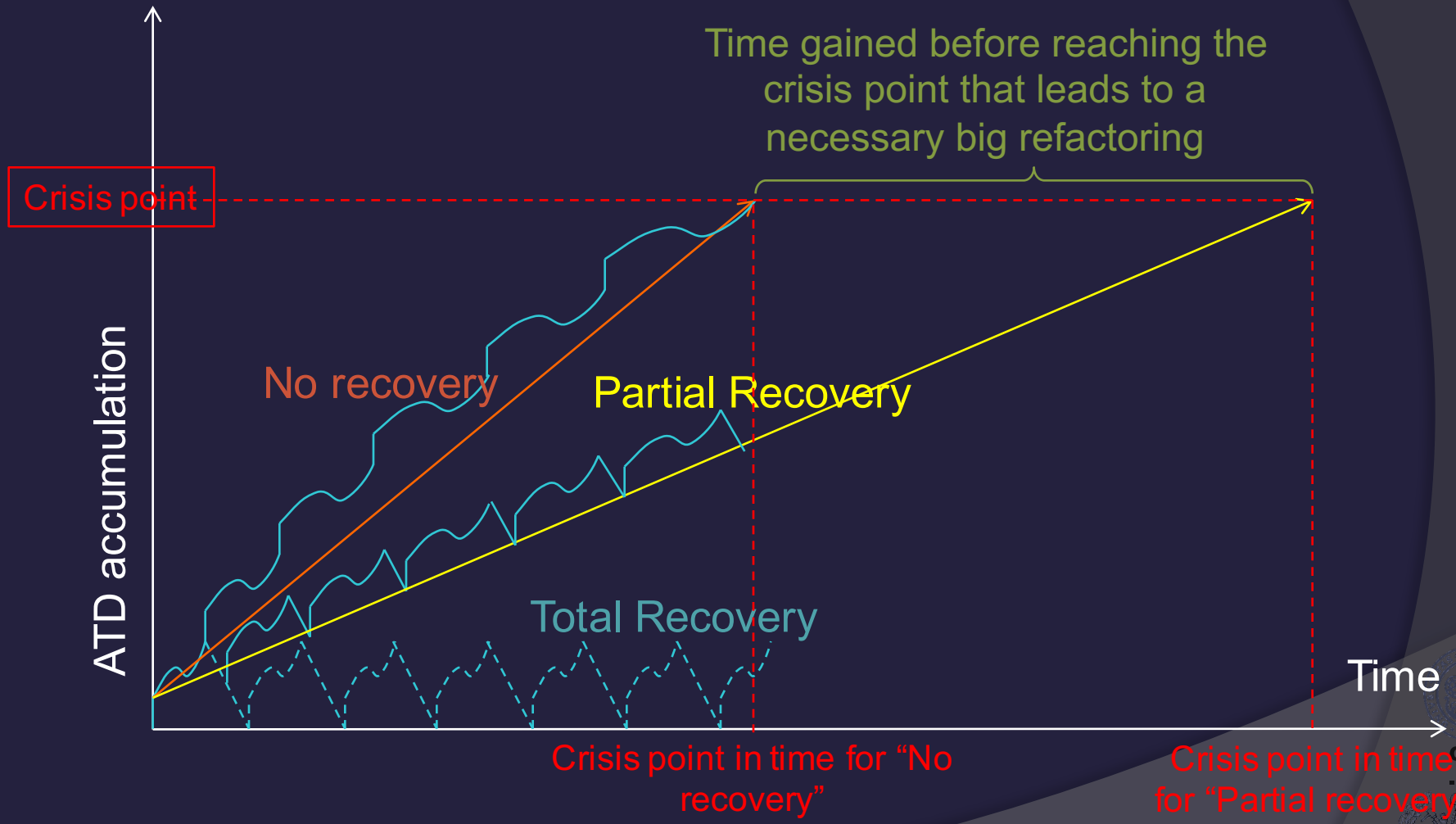
* Martini, A., Bosch, J., Chaudron, M., 2015. "*Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study*", Information and Software Technology Jounral

Accumulation and recovery over time



* Martini, A., Bosch, J., Chaudron, M., 2015. "*Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study*", Information and Software Technology Jounral

Different refactoring strategies

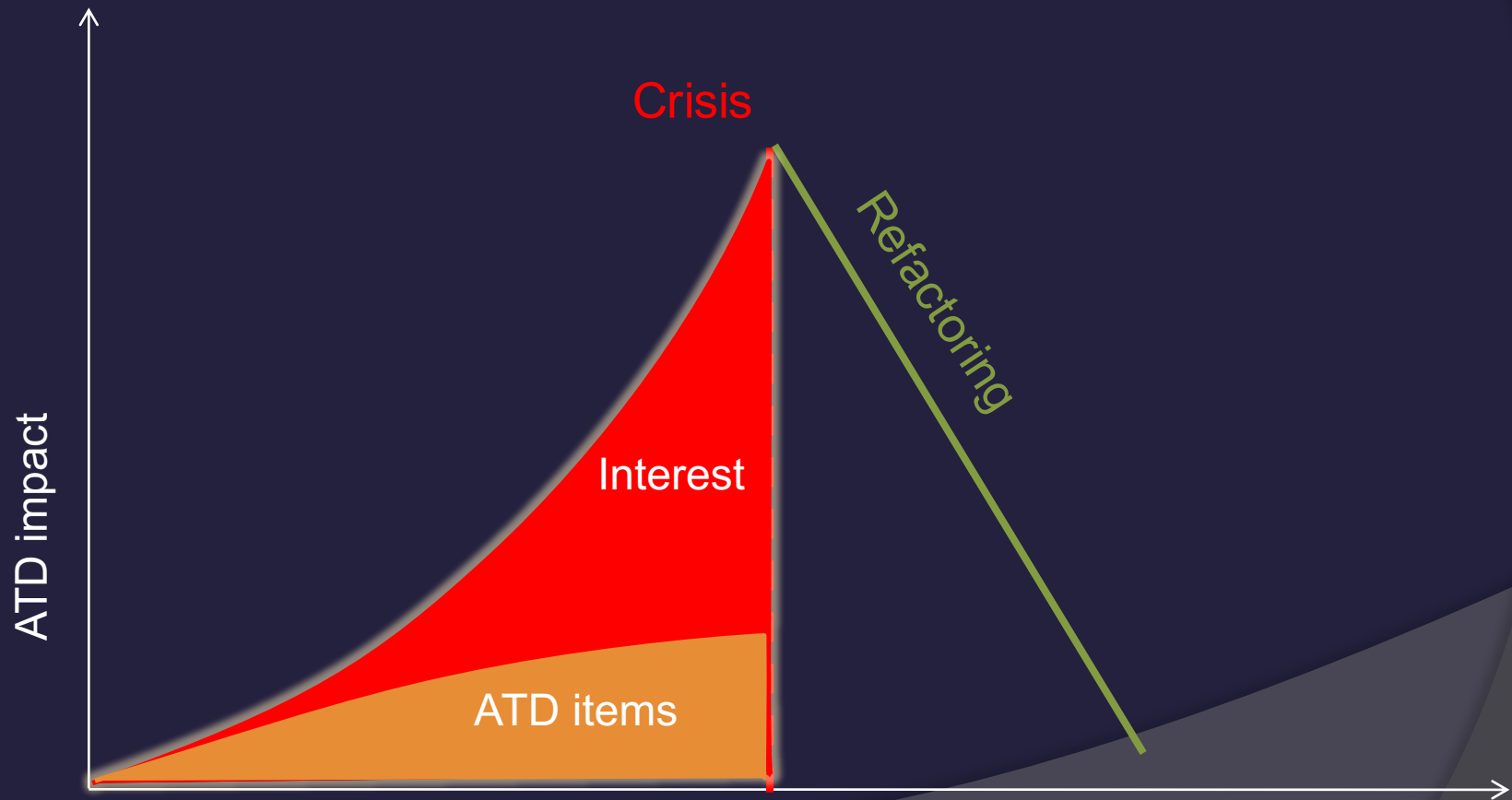


* Martini, A., Bosch, J., Chaudron, M., 2015. "*Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study*", Information and Software Technology Jounral

Consequences of Architecture Technical Debt

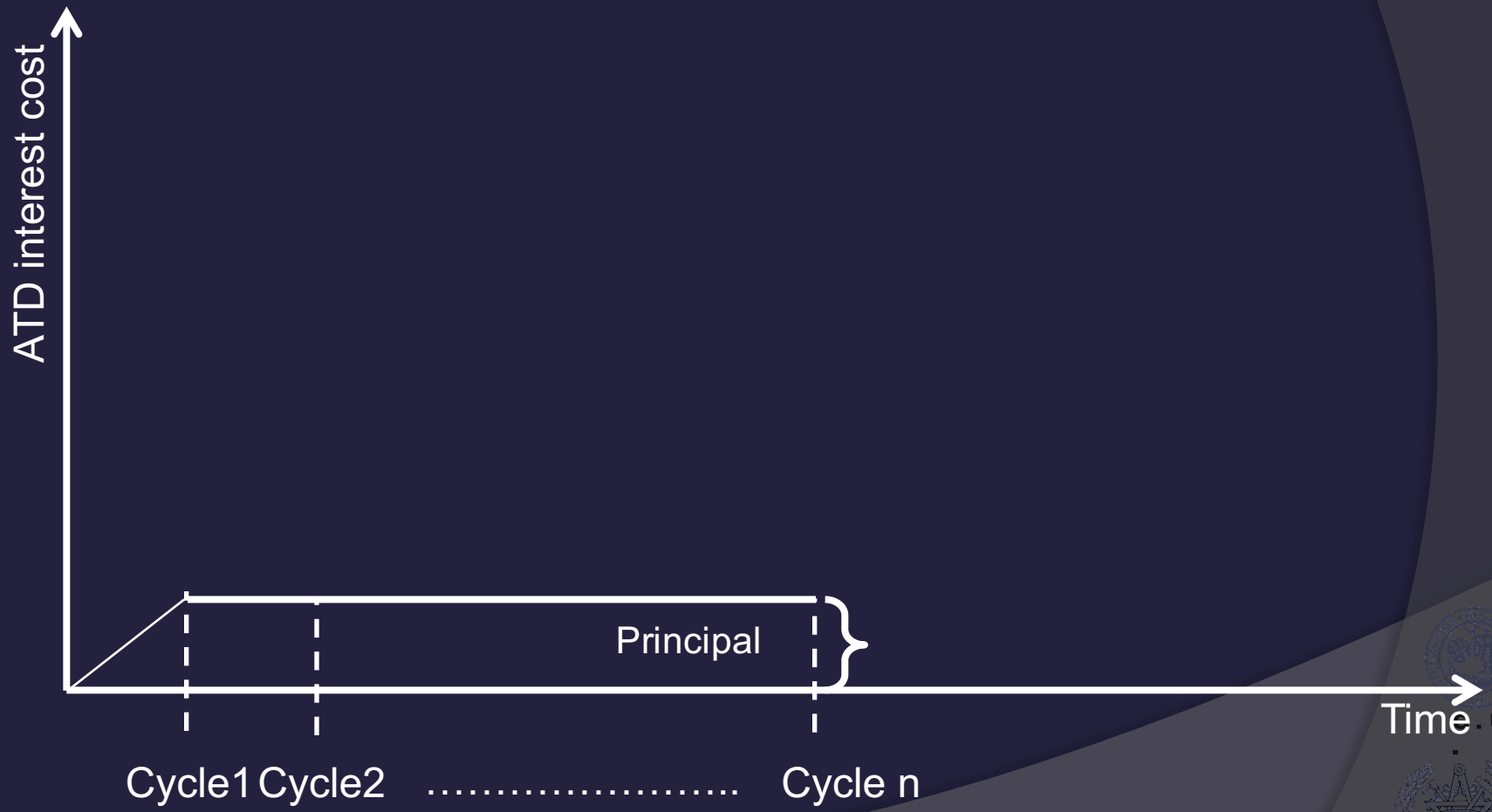
G.U.
CHALMERS

The problem is not only the accumulation of ATD but the accumulation of the interest

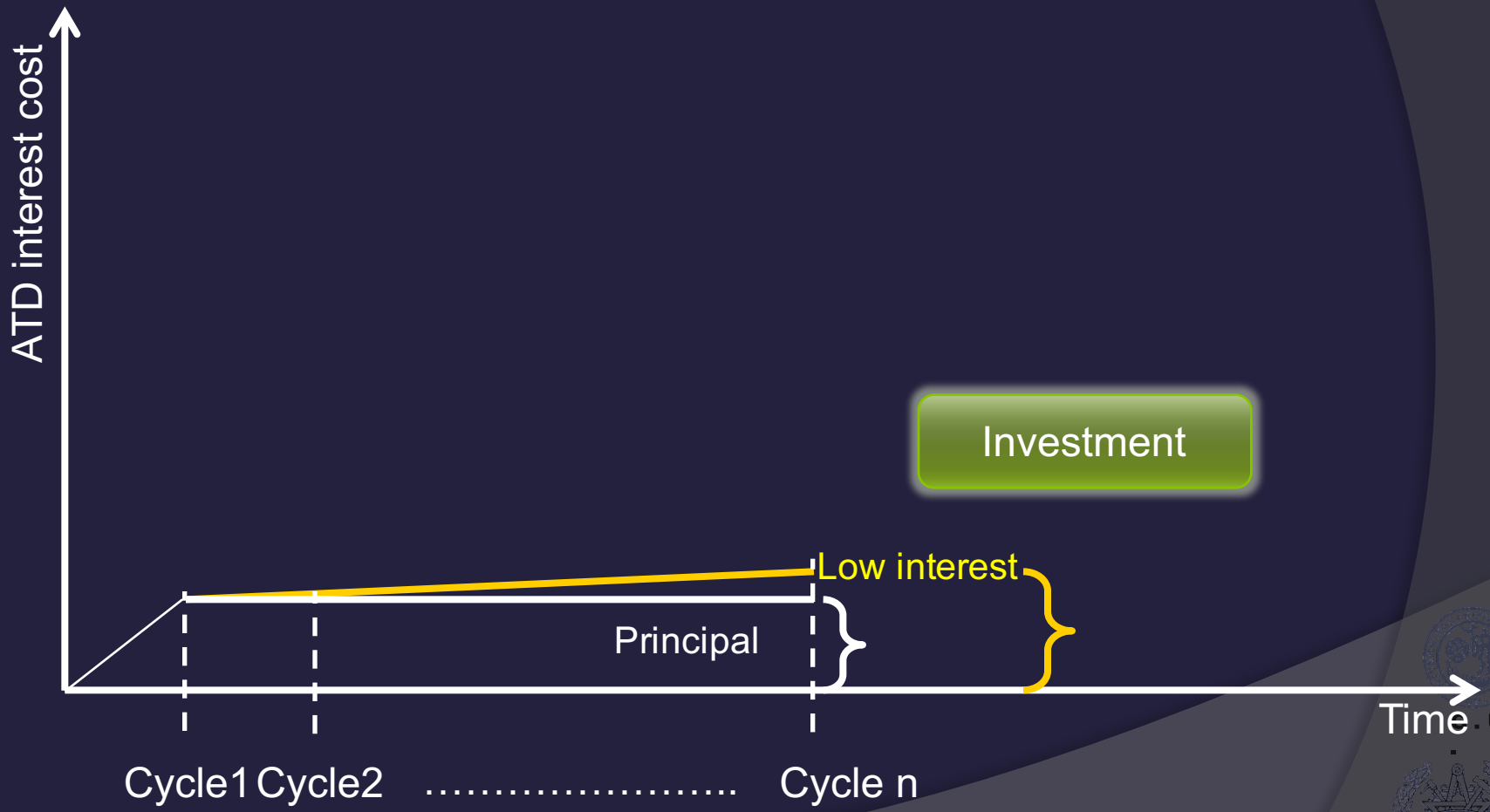


"The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles," in WICSA *2015*, Montreal, Canada.

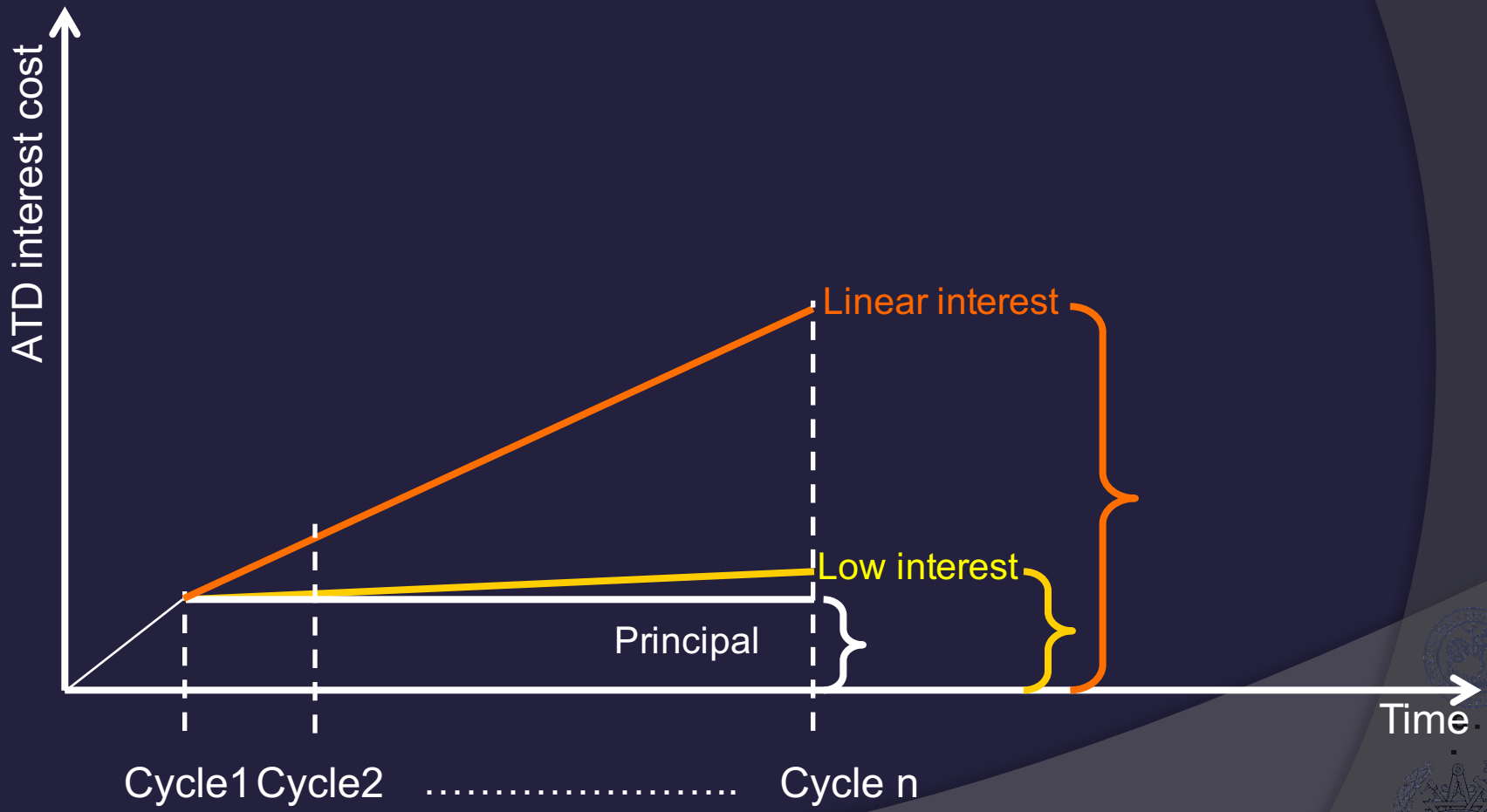
Growing interest



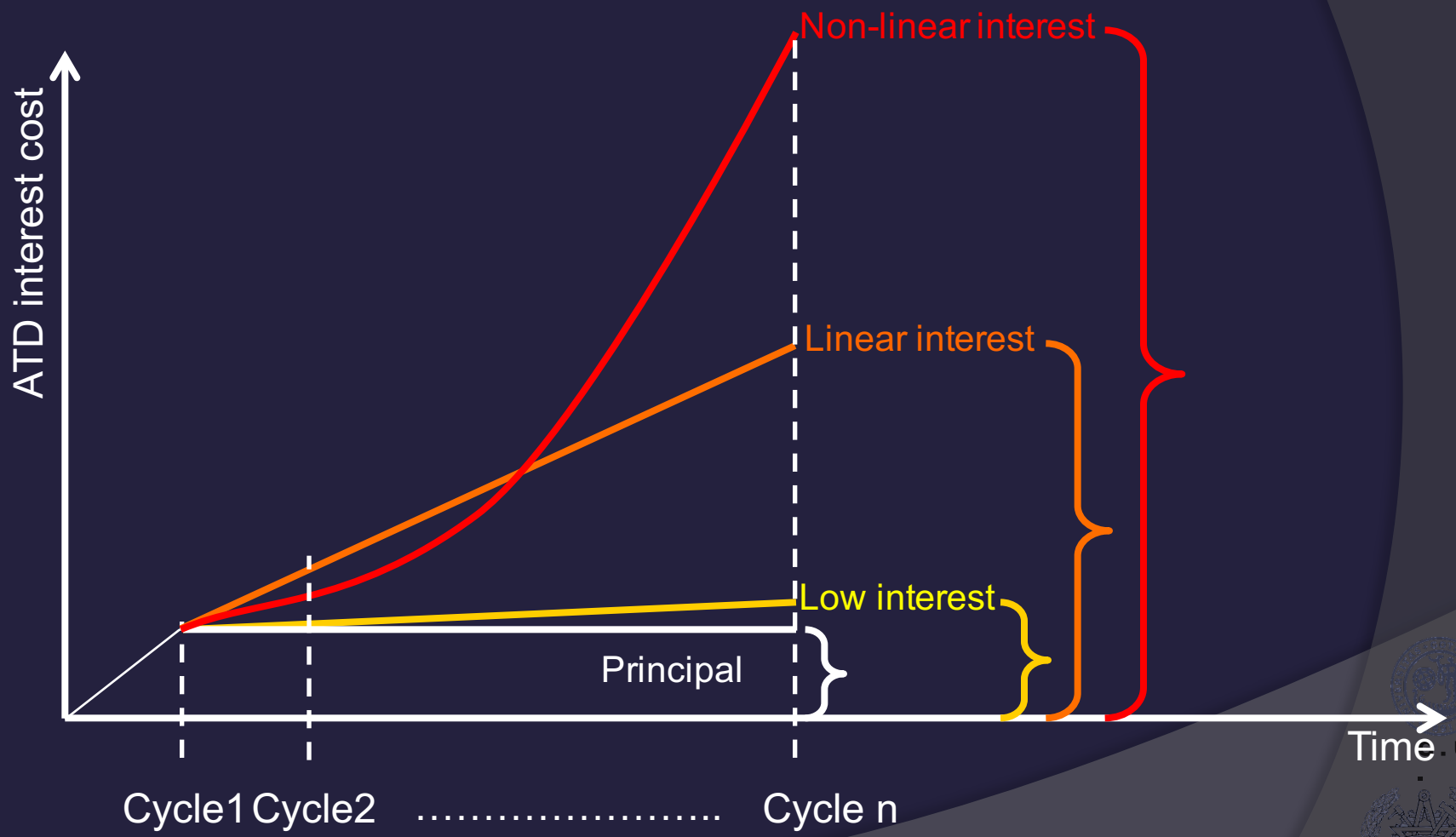
Growing interest (low)



Growing interest (linear)

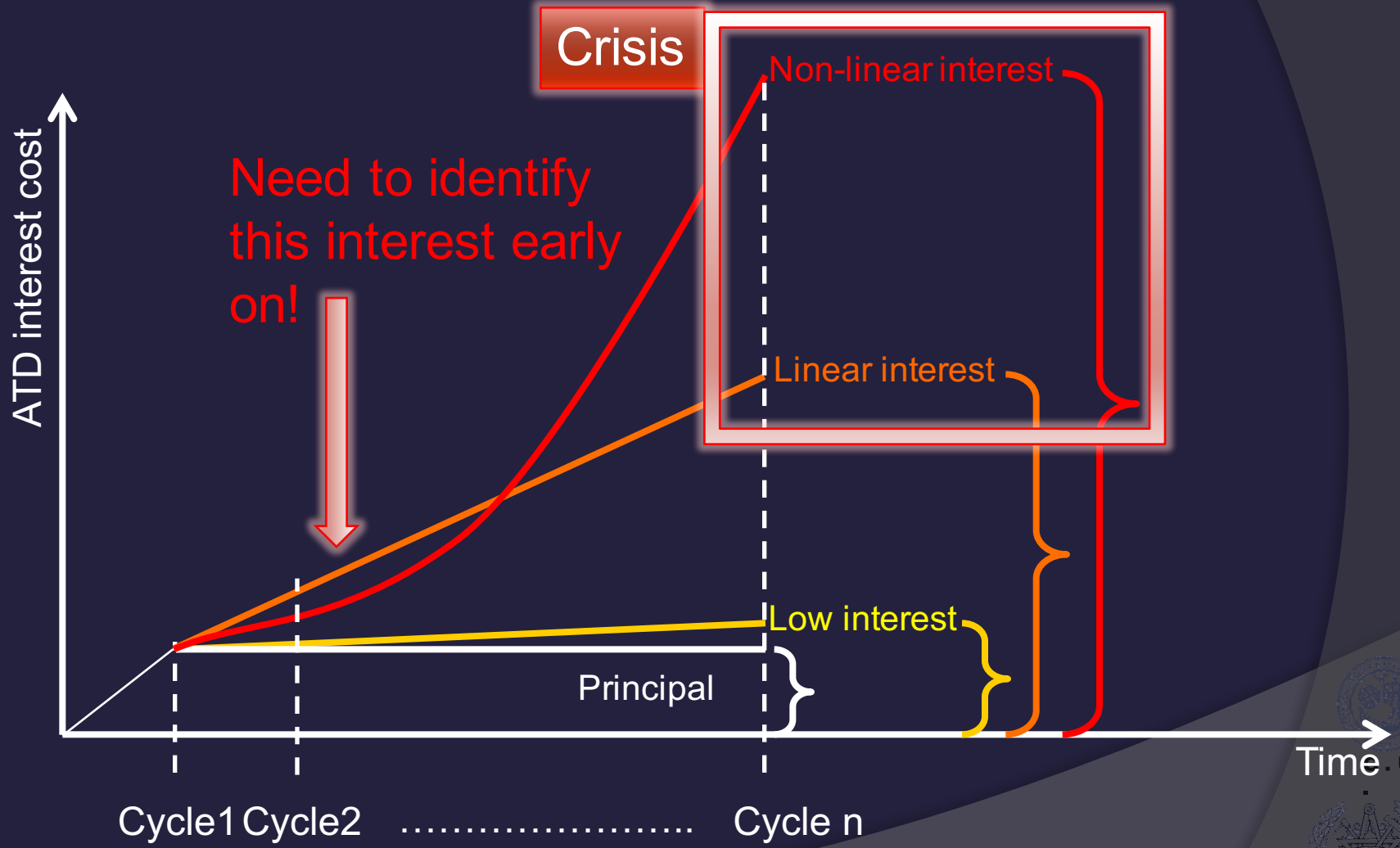


Growing interest (non linear)



CHALMERS

Growing interest to crises



"The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles," in *accepted for publication at WICSA 2015*, Montreal, Canada.

CHALMERS

We don’t want a growing interest!

What if it was the interest for the loan?

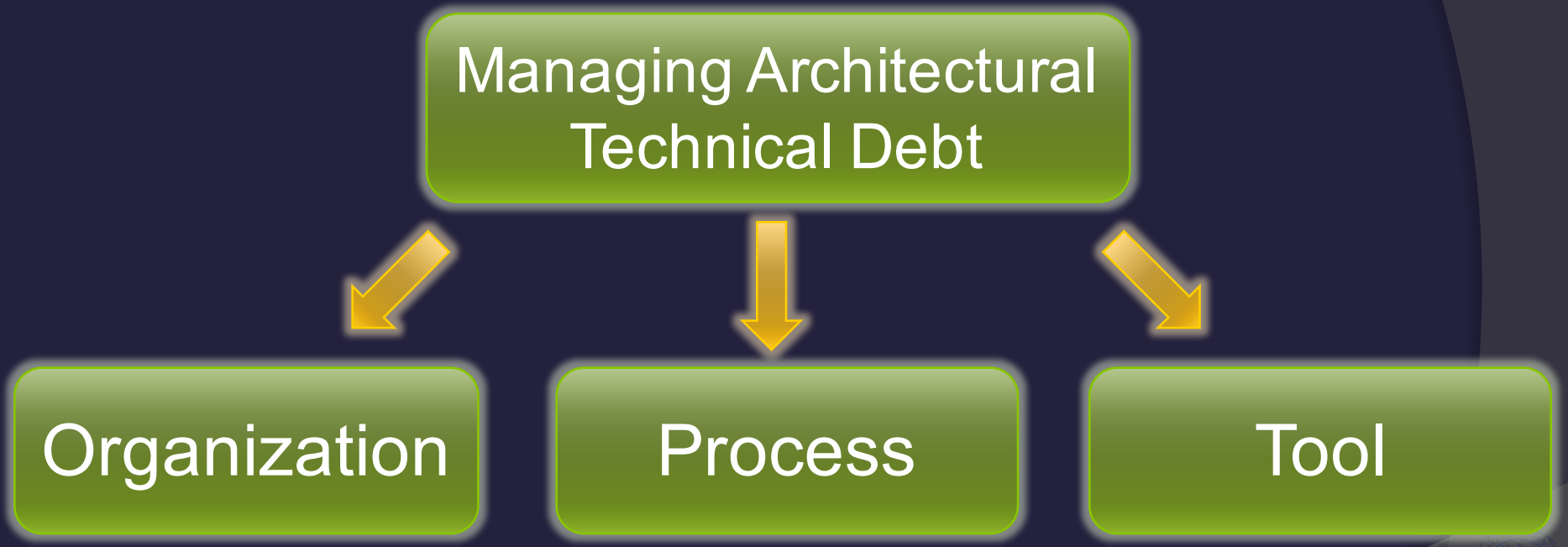
CHALMERS

Managing Architecture Technical Debt

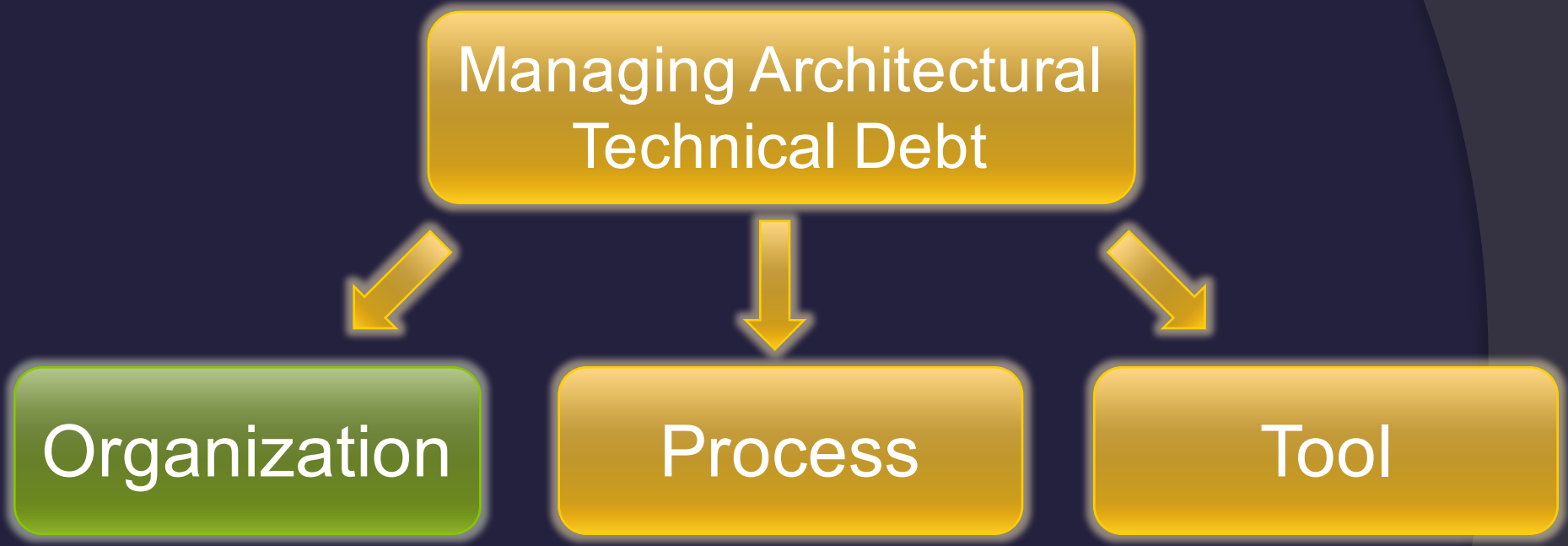
G . U .

CHALMERS

Holistic task

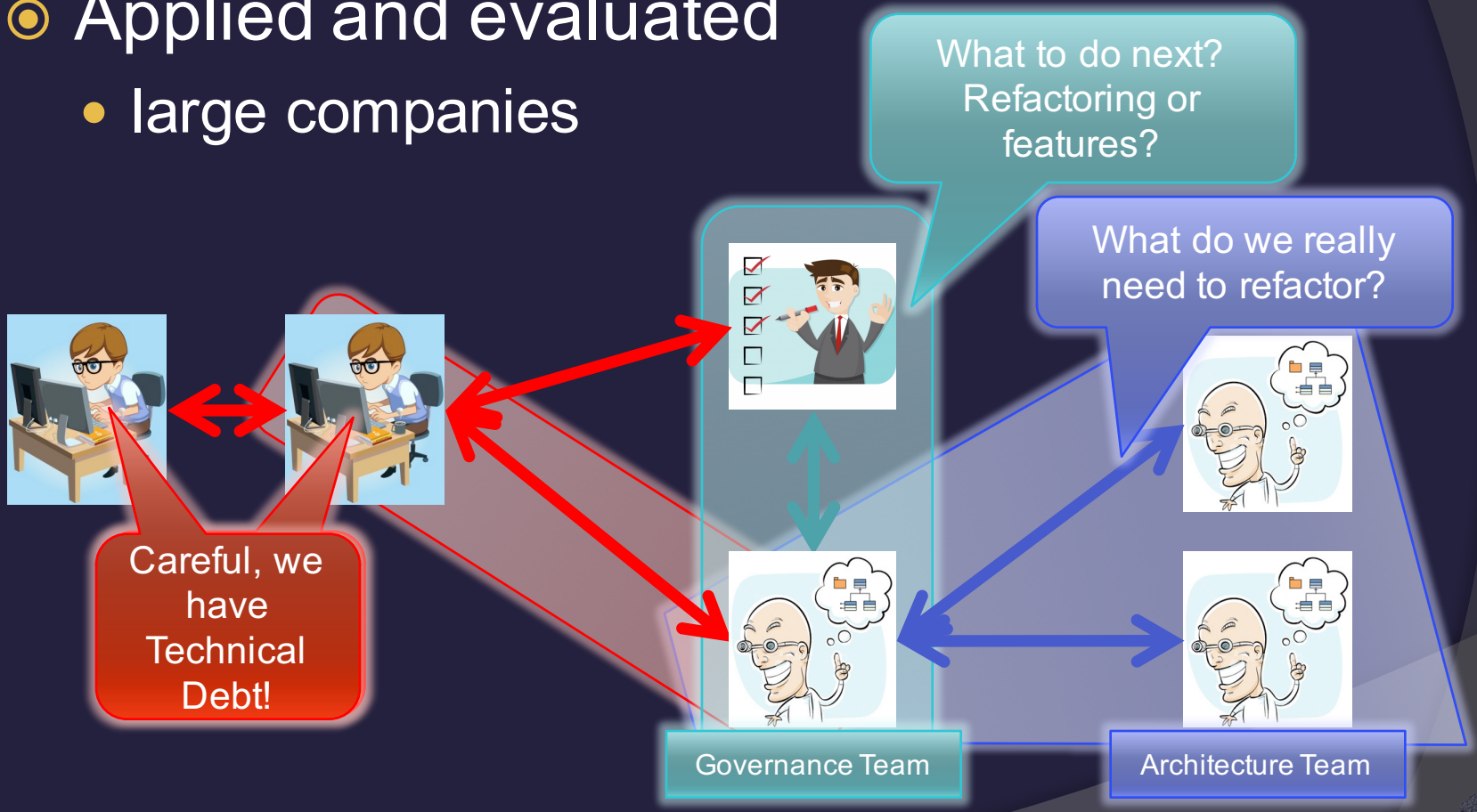


Organizational Aspect



Organization model: CAFFEA

- Applied and evaluated
  - large companies



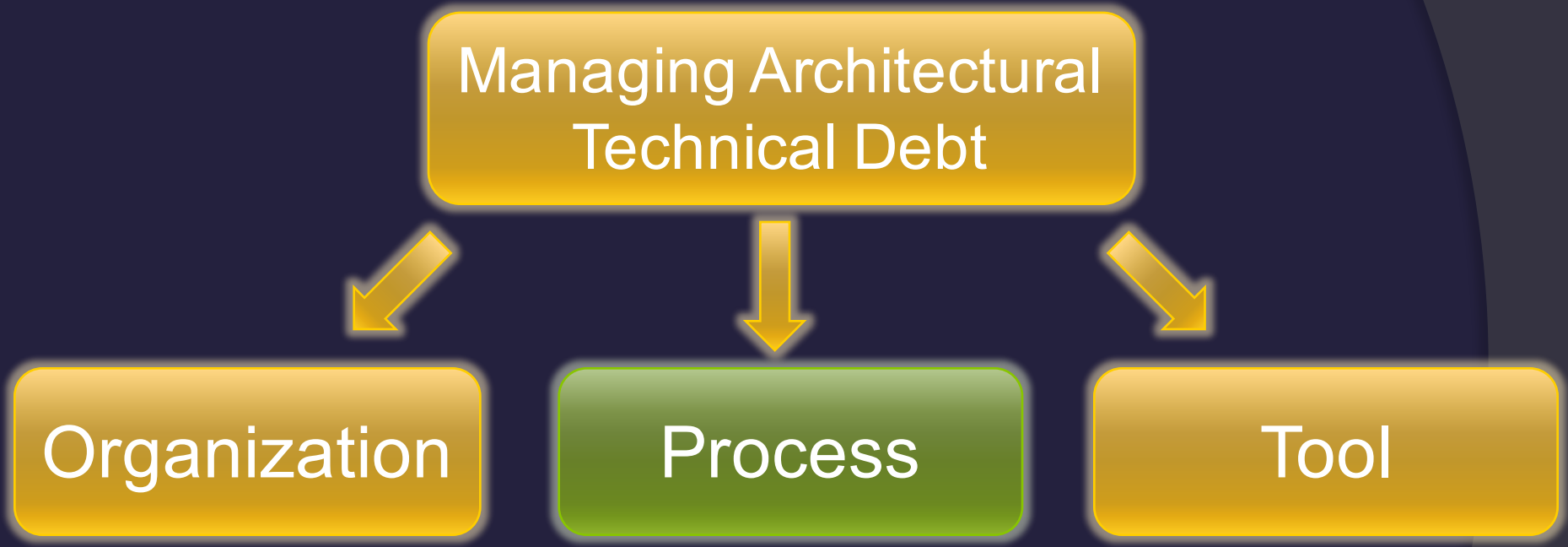
"**A Multiple Case Study of Continuous Architecting in Large Agile Companies: current gaps and the CAFFEA Framework**" in *WICSA 2016*, Venice, Italy.

CAFFEA evaluated in practice showing many improvements in:

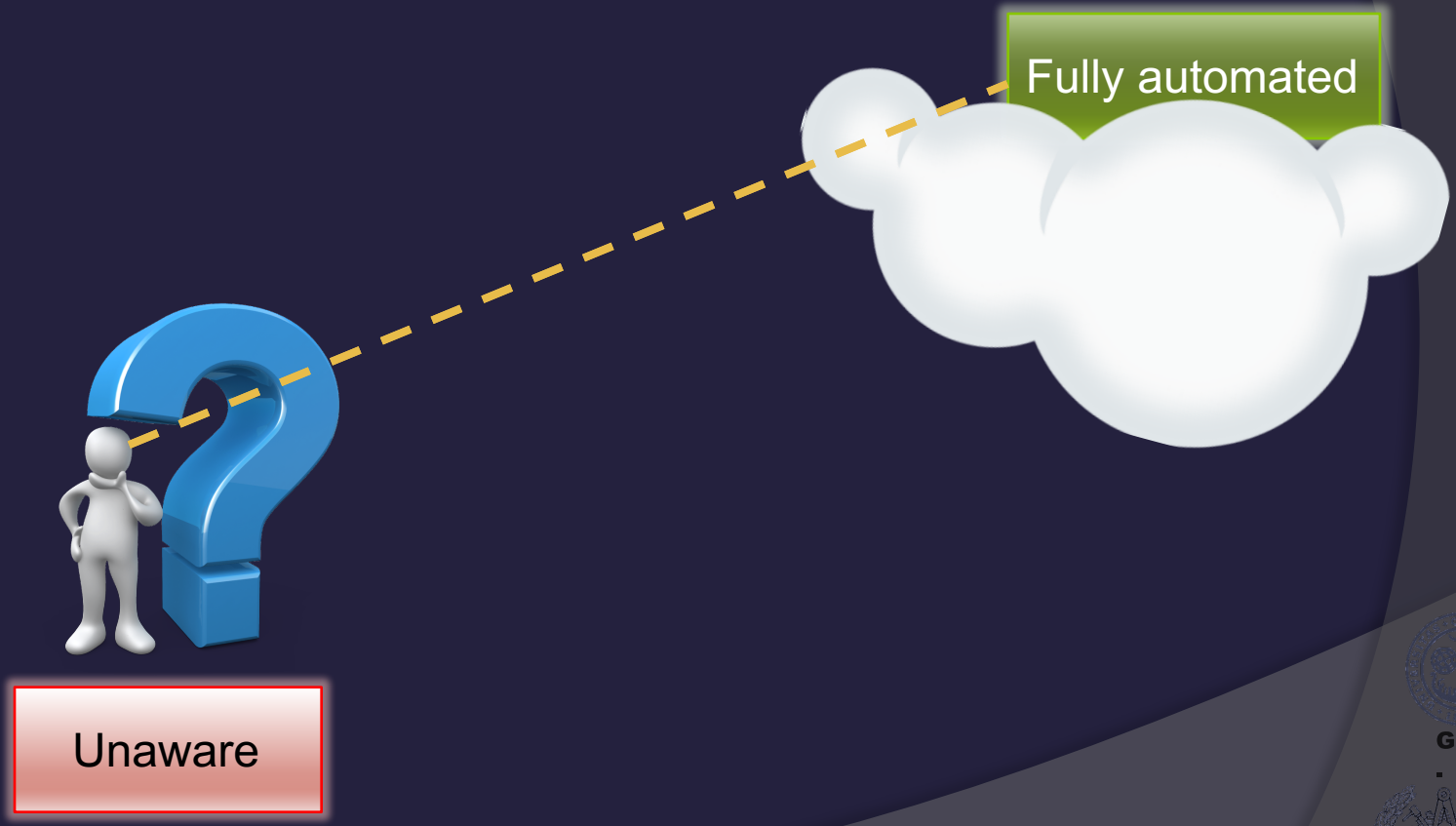
- Management of ATD
- Sharing improvements and knowledge across teams
- Tracking decisions
- Long-term perspective
- Clear references
- Monitor of architecture activities

"**A Multiple Case Study of Continuous Architecting in Large Agile Companies: current gaps and the CAFFEA Framework**" in *WICSA 2016*, Venice, Italy.

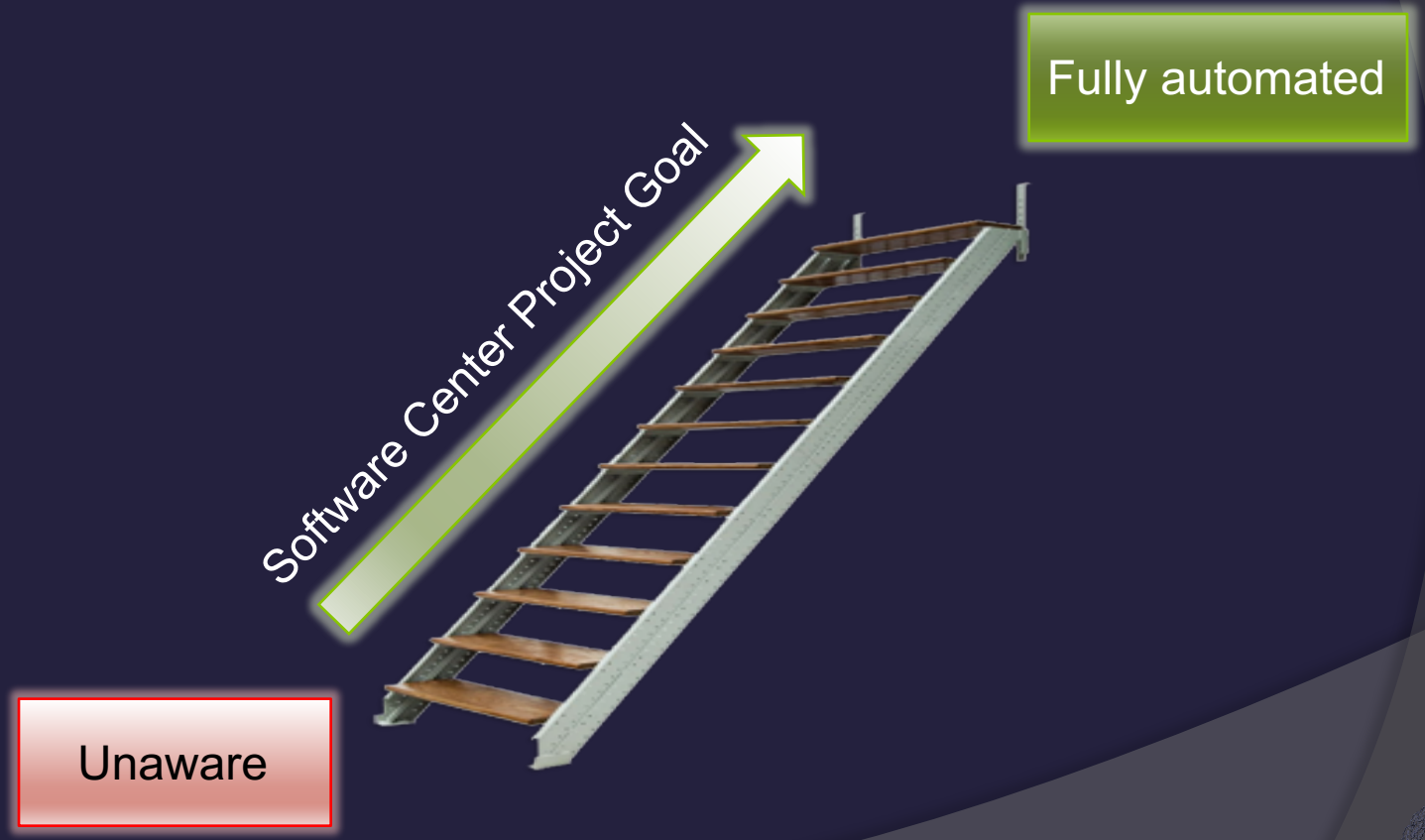
Process Aspect



How to track Technical Debt?



How to track Technical Debt?



Benefits of tracking TD

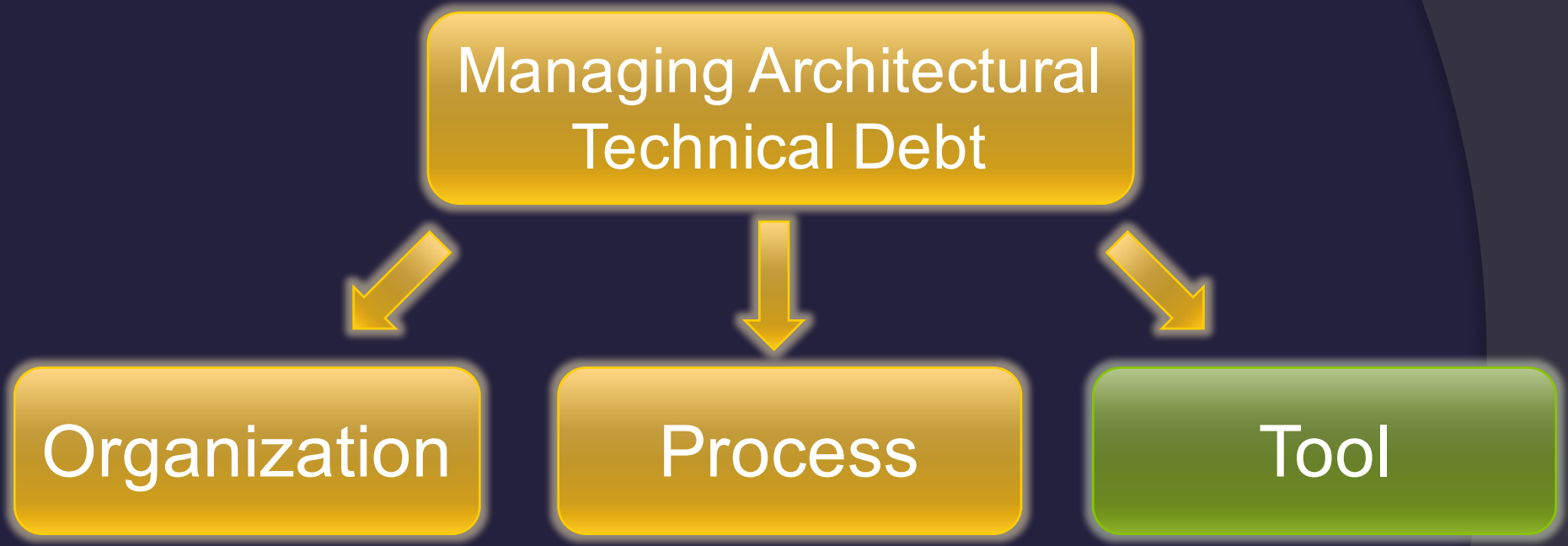
Awareness of TD

- From reactive to proactive behavior
- TD visibility
- Better estimation
- Better decisions
- Better prioritization
- Better communication

- Still some challenges, but we are working on them

Results under submission

Tool Aspect


Managing Architectural Technical Debt
Organization
Process
Tool

Should we Refactor?

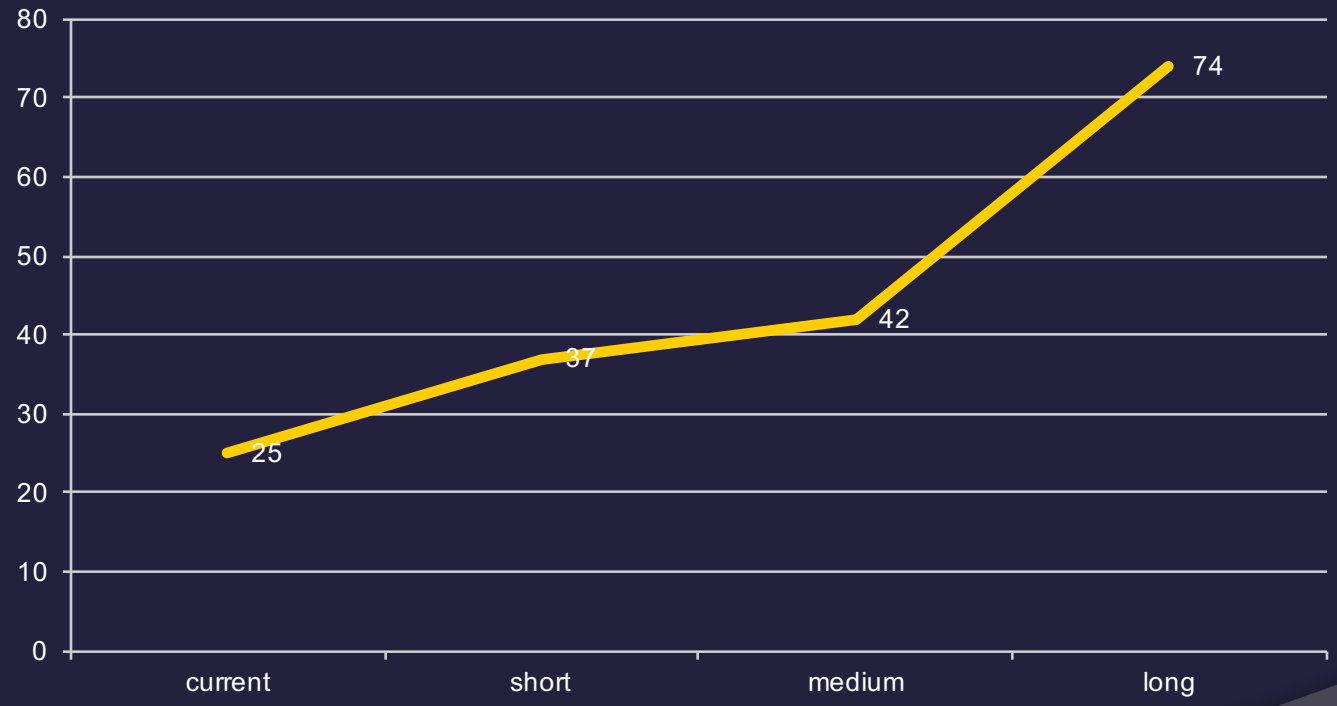
$$\frac{\text{Principal (Cost of Refactoring)}}{\text{Total Interest (Impact)}} < 1$$

When the cost of refactoring is greater than the interest, it's not convenient to refactor

A. Martini and J. Bosch, "***An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt***" ICSE SEIP 2016

Estimating the impact: AnaConDebt

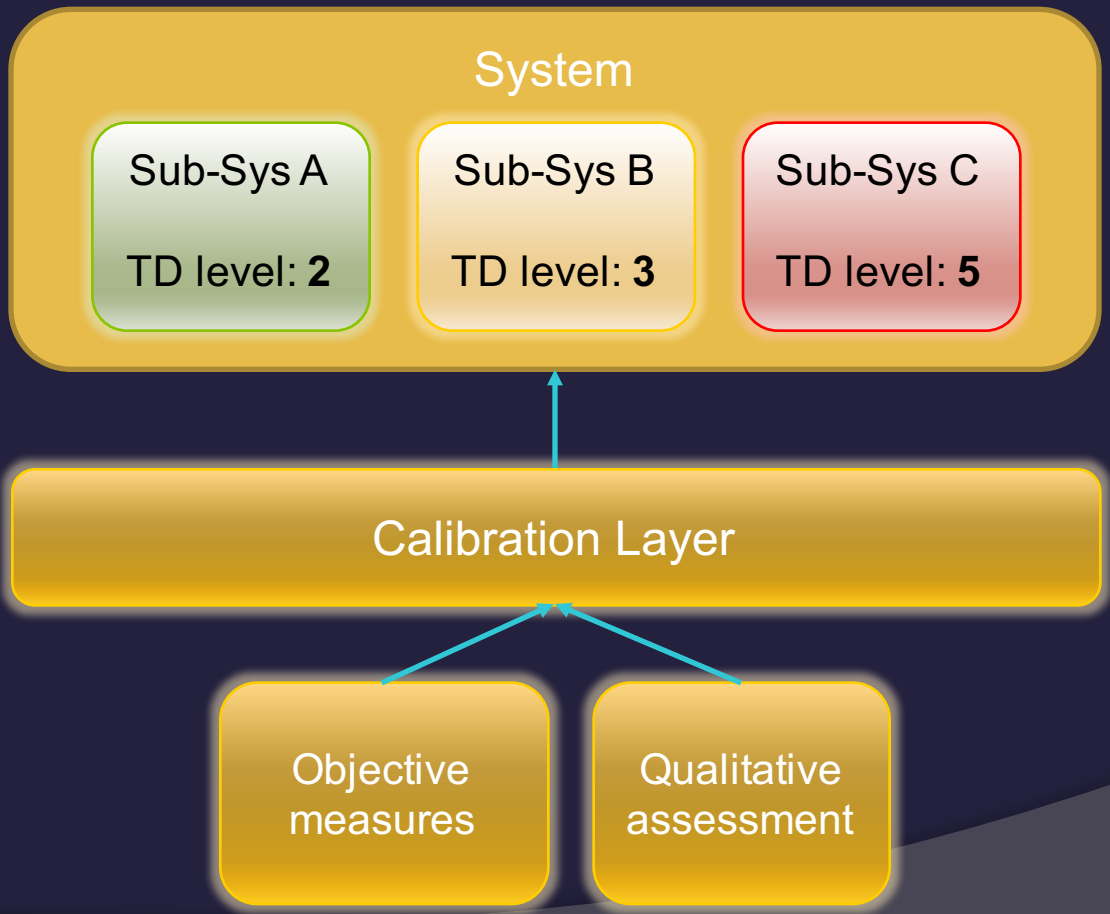
- Developed analyzing 12 cases of ATD
- Current evaluation at several companies



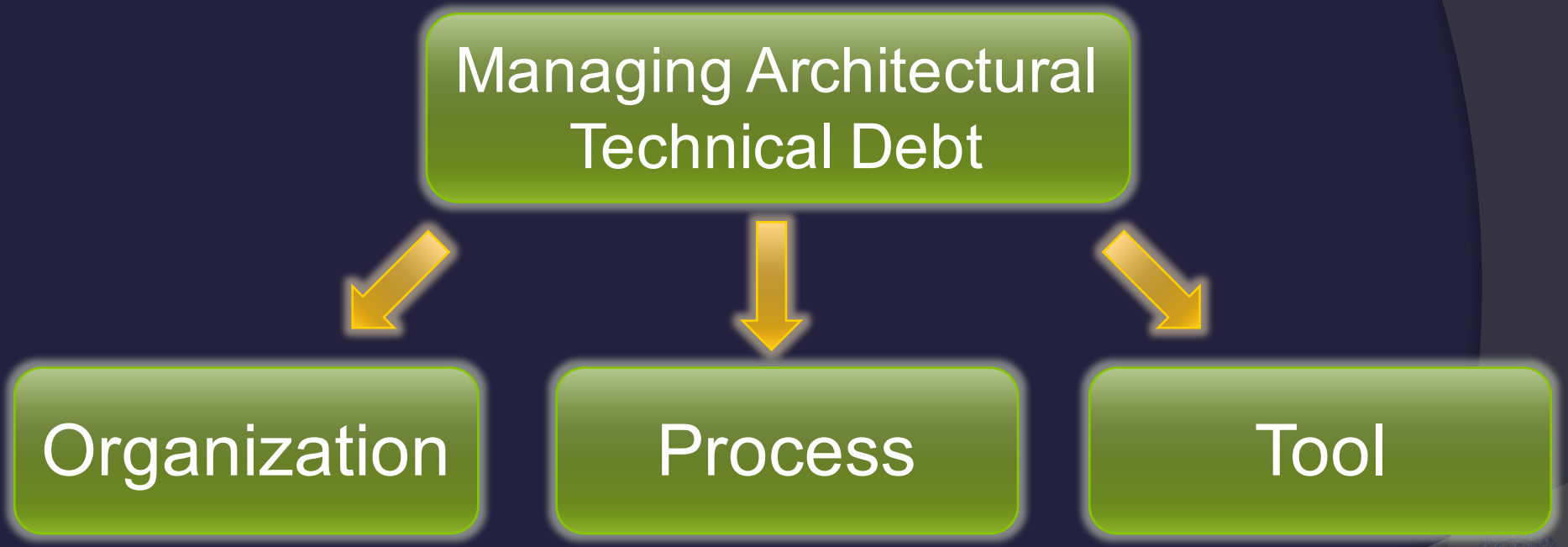
A. Martini and J. Bosch, "***An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt***" ICSE SEIP 2016

Visualization of Technical Debt

- Current consultancy work at Ericsson



Holistic goal



Take Aways

Architectural Technical Debt is inevitable

Architectural Technical Debt interest can be expensive

Architectural Technical Debt can be managed

Prioritize Technical Debt!

...to be continued...

CONTACTS

- antonio.martini@chalmers.se
- jan@janbosch.com
- https://www.linkedin.com/in/antonio-martini-79654433

PAPERS

- https://www.researchgate.net/profile/Antonio_Martini

WORK TOGETHER

- antonio.martini.am@gmail.com
- www.boschonian.com

Questions?

Comments?