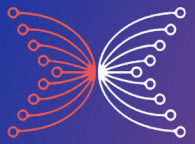
CARDANO

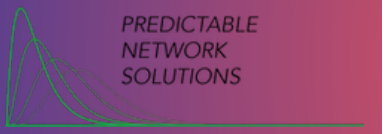
ΔQSD: Predictable Systems Design with Clearly Defined Doubt & Uncertainty

June 5th 2023 | Core Technology/Formal Methods

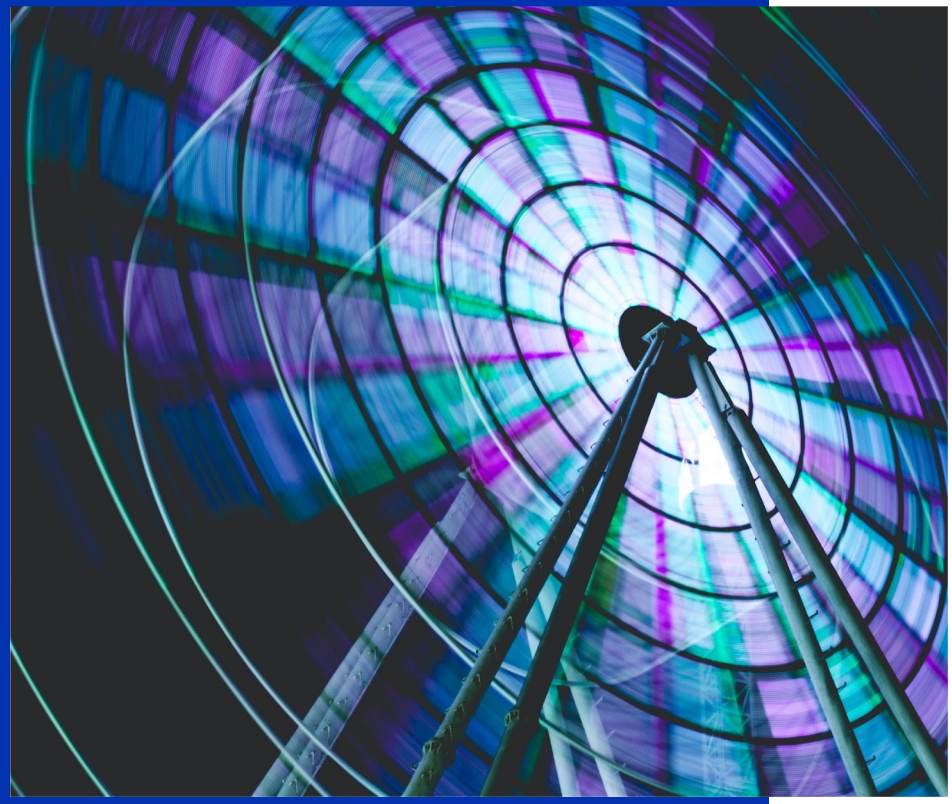
Kevin Hammond,
Neil Davies, Seyed Hossein, Peter van Roy, Peter Thompson, James Chapman



INPUT | OUTPUT



UCLouvain



Outline

01 Introduction: ΔQ Systems Design

02 Case Study 1 [Diagnosis]: iPhone Launch

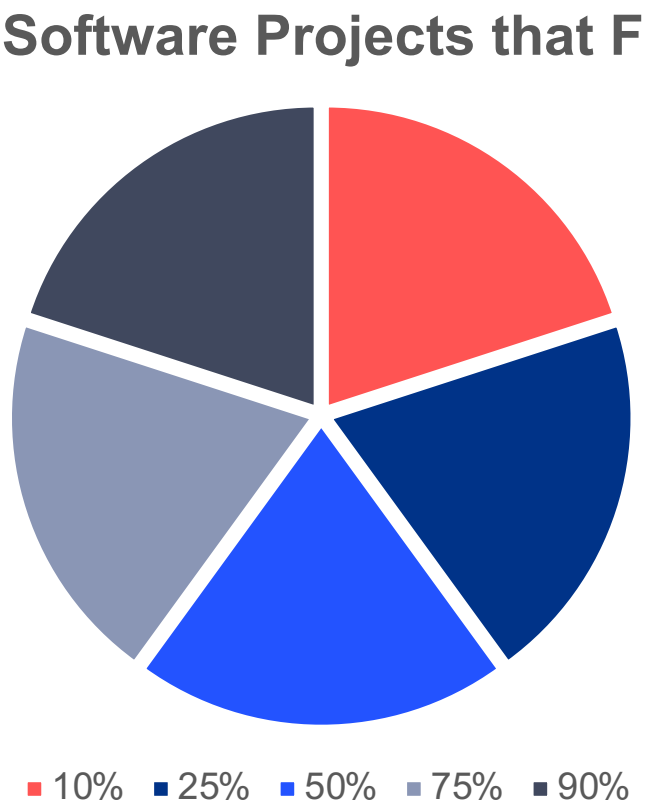
03 Case Study 2 [Design]: Blockchain Diffusion

04 Conclusions

2021 | Confidential and Proprietary

Motivation

Audience Survey



Overall 68% of Software Projects Fail

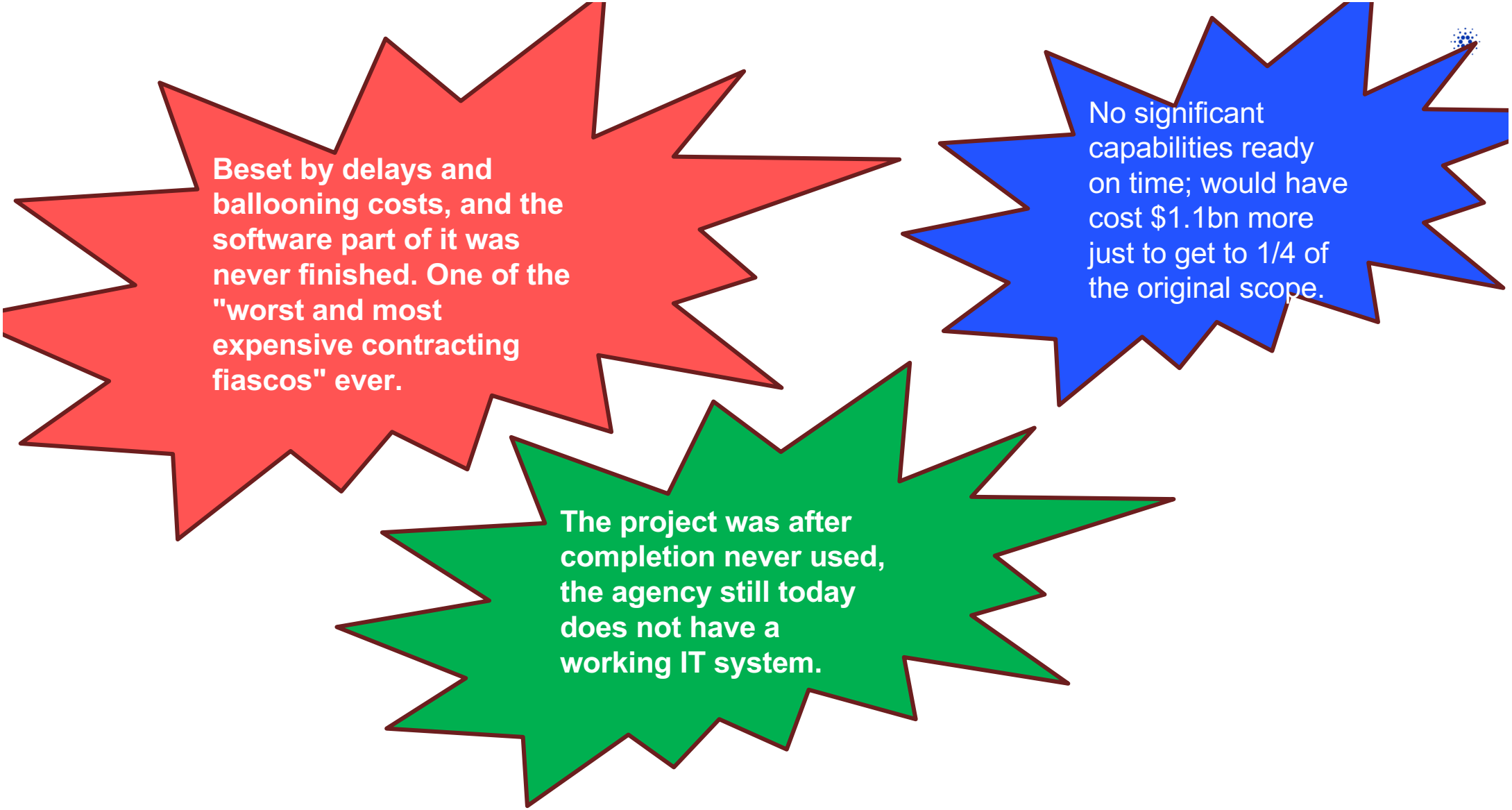
- Large projects are much more likely to fail than small one
  - 90% of large projects are failures
- Poor requirements analysis incurs a 60% cost premium
- Agile techniques don’t usually help much, if at all
- Fixing a problem in development costs 100x fixing it in design
- $260bn/annum cost for failed projects


The World’s Most Expensive
Software Failures?

	Started	Termin-ated	Name	Type of System	Country		Cost (expected)	In House or Outsourced	Outcome
1.	2002	2011	NHS Connecting for Health	Electronic care records	United Kingdom		£12bn *(£2.3bn)*	Outsourced	Discontinued, but some parts continued
2.	1982	1994	FAA Advanced Automation System	Air Traffic Control	United States		$3–6b	?	Scrapped
3.	2005	2012	Expeditionary Combat Support System	Military Enterprise Resource Planning	United States		$1.1bn	Outsourced – including requirements	Cancelled
4.	2007	2010	Försäkringskassan SAP	Dental health service system	Sweden		SEK 10bn	Outsourced, then insourced	Cancelled
5.	1982	1994	FAA Advanced Automation System	Air Traffic Control	United States		$3–6b	?	Scrapped
6.	2000	2009	Customer Account Data Engine	System for handling tax records	United States		~US$500 million	Outsourced to IBM, Northrop Grumman and others	Abandoned, intended to be replaced by CADE 2
7.	2007	2014	e-Borders	Advanced passenger information programme	United Kingdom		over £412m (£742m)	Outsourced	Cancelled

Source: Wikipedia


Beset by delays and ballooning costs, and the software part of it was never finished. One of the "worst and most expensive contracting fiascos" ever.
No significant capabilities ready on time; would have cost $1.1bn more just to get to 1/4 of the original scope.
The project was after completion never used, the agency still today does not have a working IT system.

Some Key Problems with Modern Software Development Methodologies

1. **Emphasise rapid and flexible software construction**
   - **Rather than careful design**
2. **Fail to adequately consider essential quality requirements**
3. **Fail to consider properly whether a system can actually meet its intended outcomes**
4. **Fail to consider deployment at scale**

We Need Good Design Exploration & System Diagnosis

CARDANO

Don’t waste time building **unworkable** software systems

- is a complex system feasible?
- will the system perform as needed?
- how can the design be refined?
- how do new requirements change a system?
- why is the system not working as expected?

How Functional Programming Can Help

Functional programming techniques provide

- **composability**
- **structure**
- **reasonability**

They allow us to cleanly isolate different concerns

“We demand rigidly defined areas of doubt and uncertainty!”
Vroomfondel, The Hitchikers Guide to the Galaxy


INPUT | OUTPUT
The ΔQ System Design Paradigm

The ΔQSD System Design Paradigm

ΔQSD is an industrial-strength paradigm for system design exploration

- **Allows early prediction of performance and feasibility for complex distributed systems**
- **Developed over 30 years by Predictable Network Solutions Ltd. (PNSol) & others**
- **Widely used and validated in large industrial projects**
  - **Large cumulative savings in project costs**

"Mind Your Outcomes", Computers 2022, 11, 45
https://www.mdpi.com/2073-431X/11/3/45

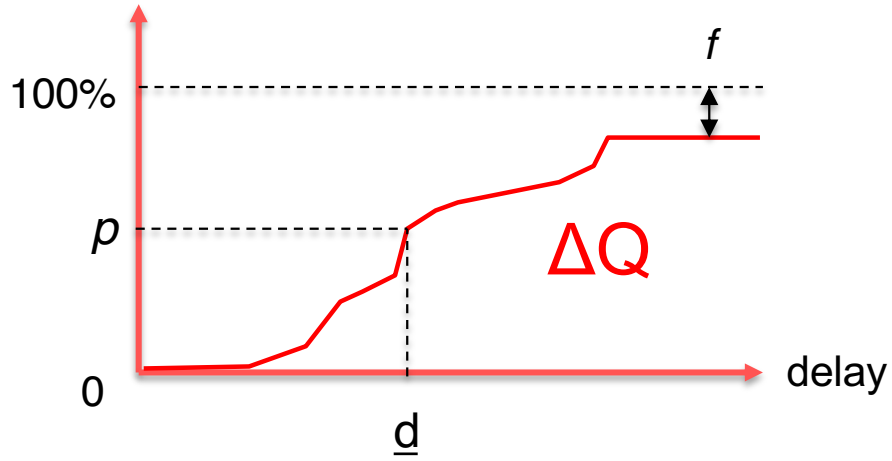
ΔQSD Properties

- **Compositional: first-class latency and failure**
- **Stochastic approach to capture uncertainty in the design**
- **Performance (latency and throughput) and feasibility can be predicted at high system load for partially defined systems**
- **Dependencies and multiple timescales are added to the compositional approach**

Key Concept 1: Quality Attenuation

Quality attenuation (ΔQ): "first-class latency and failure"

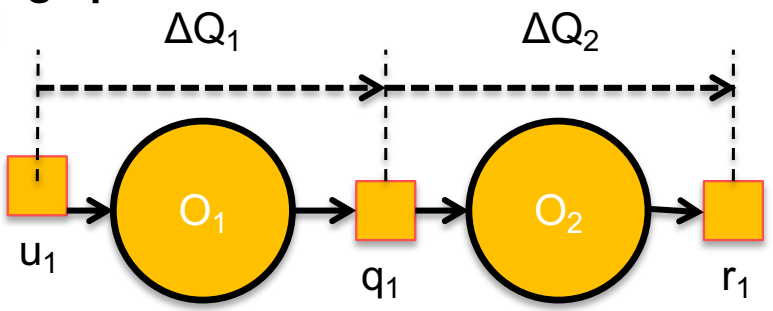
- **A ΔQ is a cumulative distribution function**
  - **defines both latency and failure probability between start and end event**
- **Since ΔQ combines latency and failure => easy to examine latency/failure trade-off**



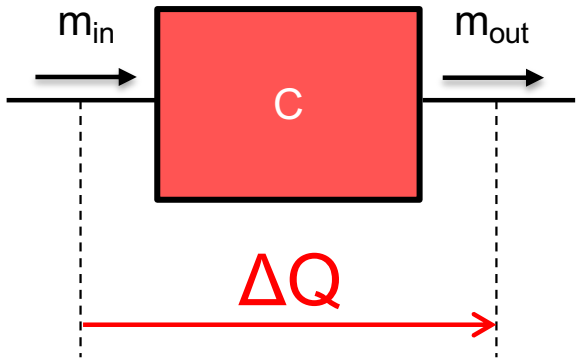
Key Concept 2: Outcome Diagram

Outcome diagram: "system observed from outside"

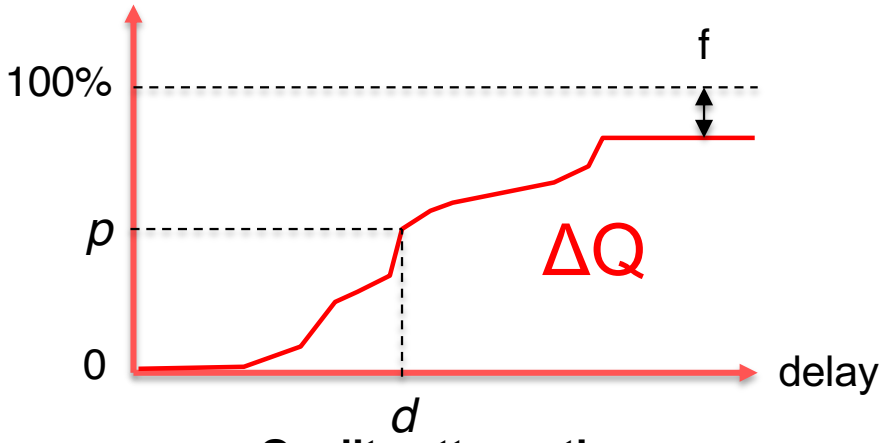
- An outcome is any well-defined system behaviour with observable start and end events
  - each outcome has a ΔQ
- An outcome diagram is a causal directed graph
  - defines relationships between all system outcomes
  - ΔQ can be calculated for the system as a whole
- The outcome diagram can be used during the whole design process. It can express partially defined systems that are refined from an initial unknown design up to the final constructed system



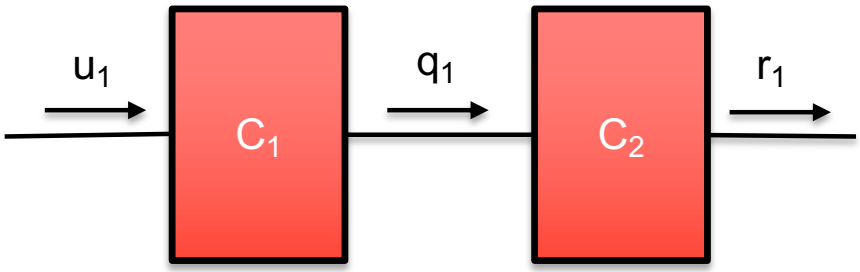
Example Quality Attenuation and Outcome Diagram



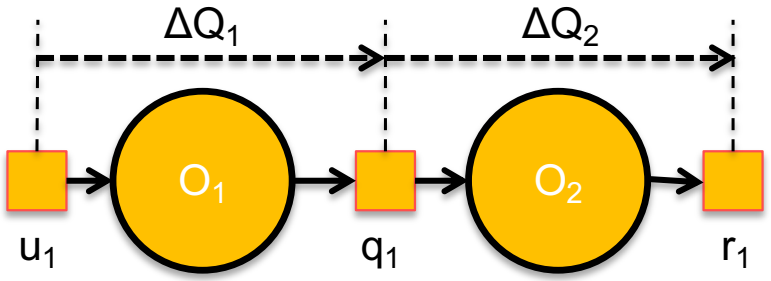
System component



Quality attenuation

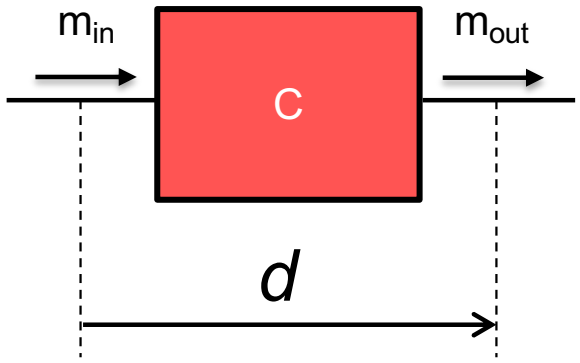


System block diagram

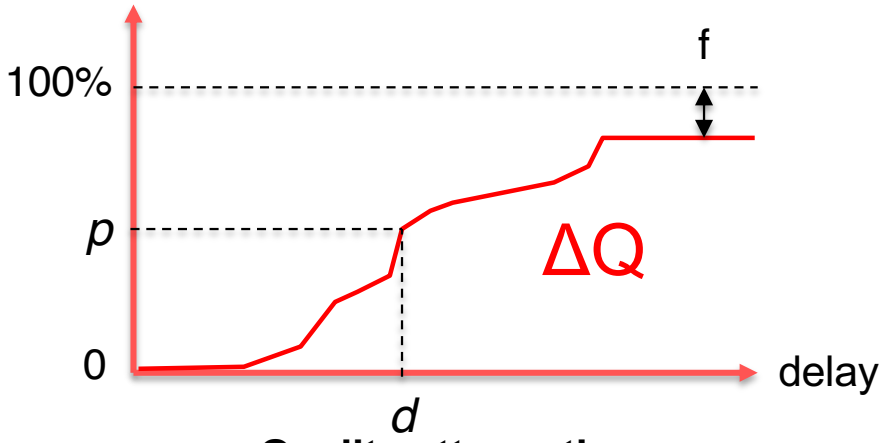


Outcome diagram

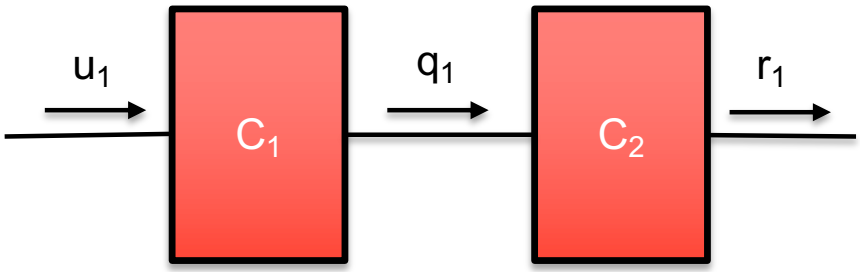
Example Quality Attenuation and Outcome Diagram



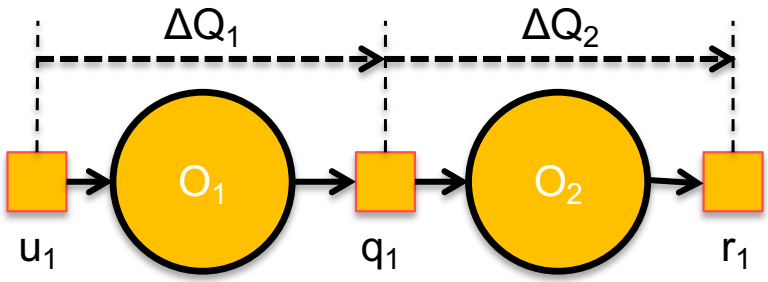
System component



Quality attenuation



System block diagram



Outcome diagram

Using ΔQ

We combine ΔQ_i of components C_i to get the ΔQ_{system} of the whole system

- **If there is something wrong with ΔQ_{system} then we reason backwards to pinpoint the problem**

Diagnosis Versus Design

ΔQSD can be used in two ways

1. For Diagnosis

- **debugging existing systems with problems**

2. For Design: designing systems using ΔQSD from the start

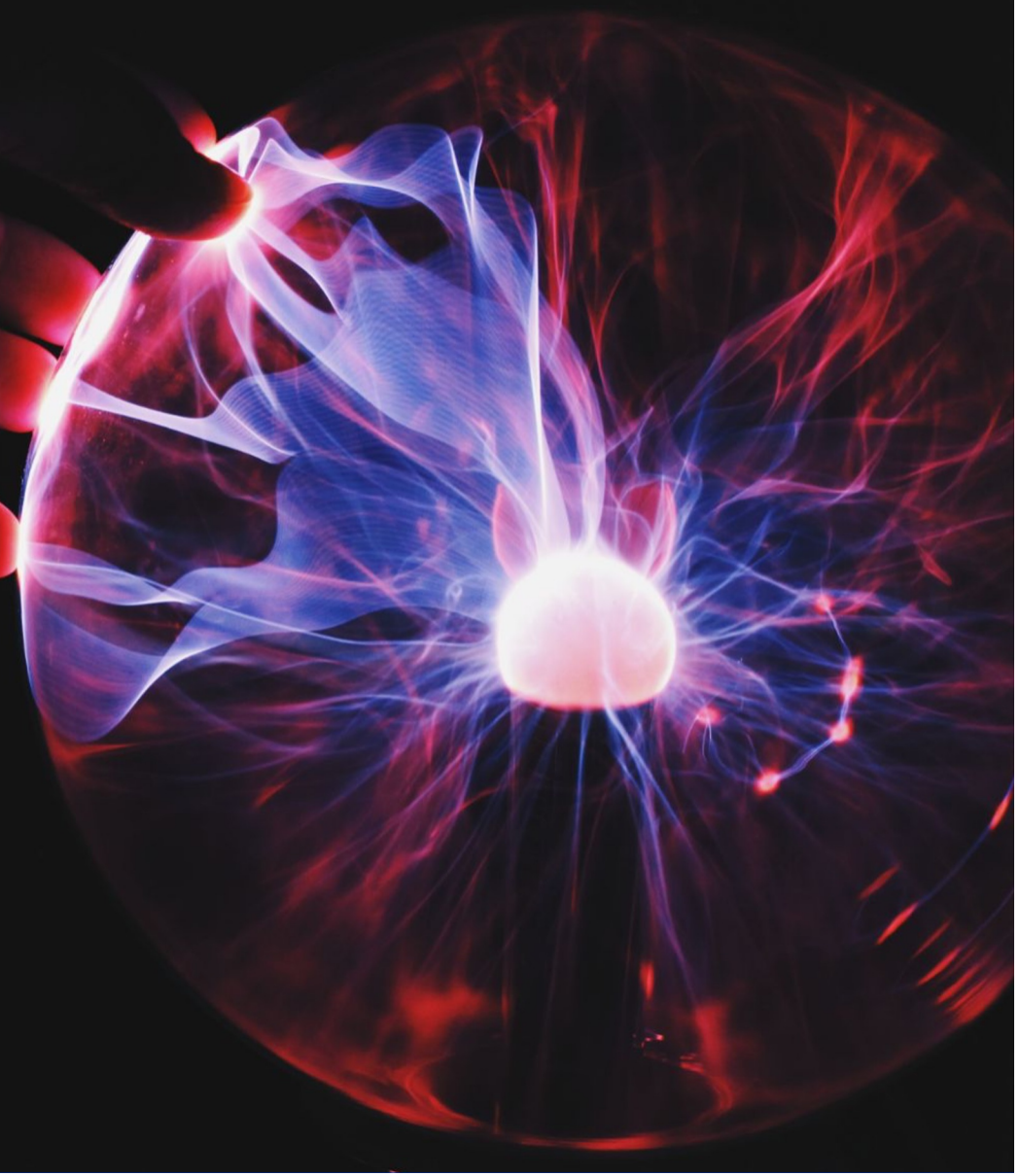
It's better to use ΔQSD for design rather than diagnosis

- **Prevention is better than cure!**

INPUT | OUTPUT

Case Study 1: iPhone Launch

(Diagnosis)



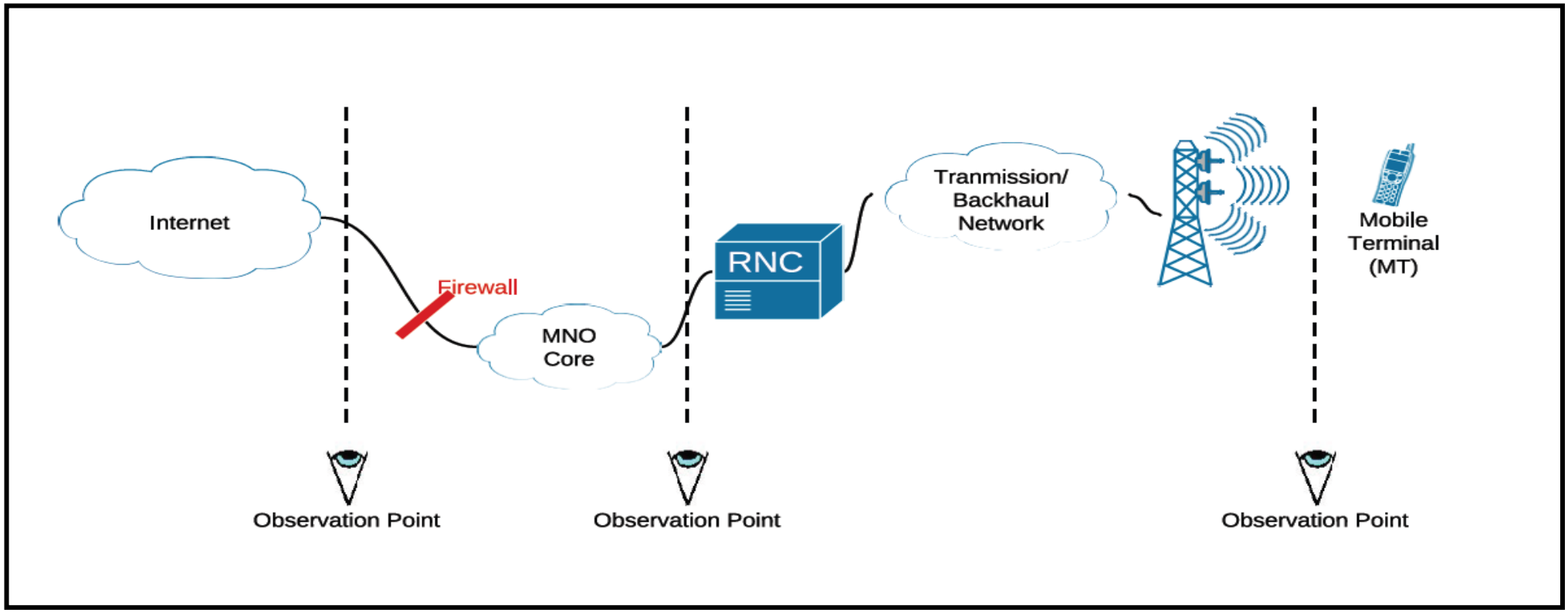
iPhone launch case study

iPhone was initially supported in UK by one Mobile Network Operator (MNO)

A second MNO prepared to enter this market

- **Before the launch, the performance was known to be bad for MNO #2**
  - **MNO #1 had gleefully prepared a major ad campaign focusing on this**
- **Using ΔQSD, PNSol diagnosed and corrected the problem just before launch**
  - **Thus saving the bacon of MNO #2**
- **Result was a 100% improvement in `http` download KPI**
  - **This placed MNO #2 in first place**
  - **To the great embarrassment of MNO #1**

Diagnosis approach and solution



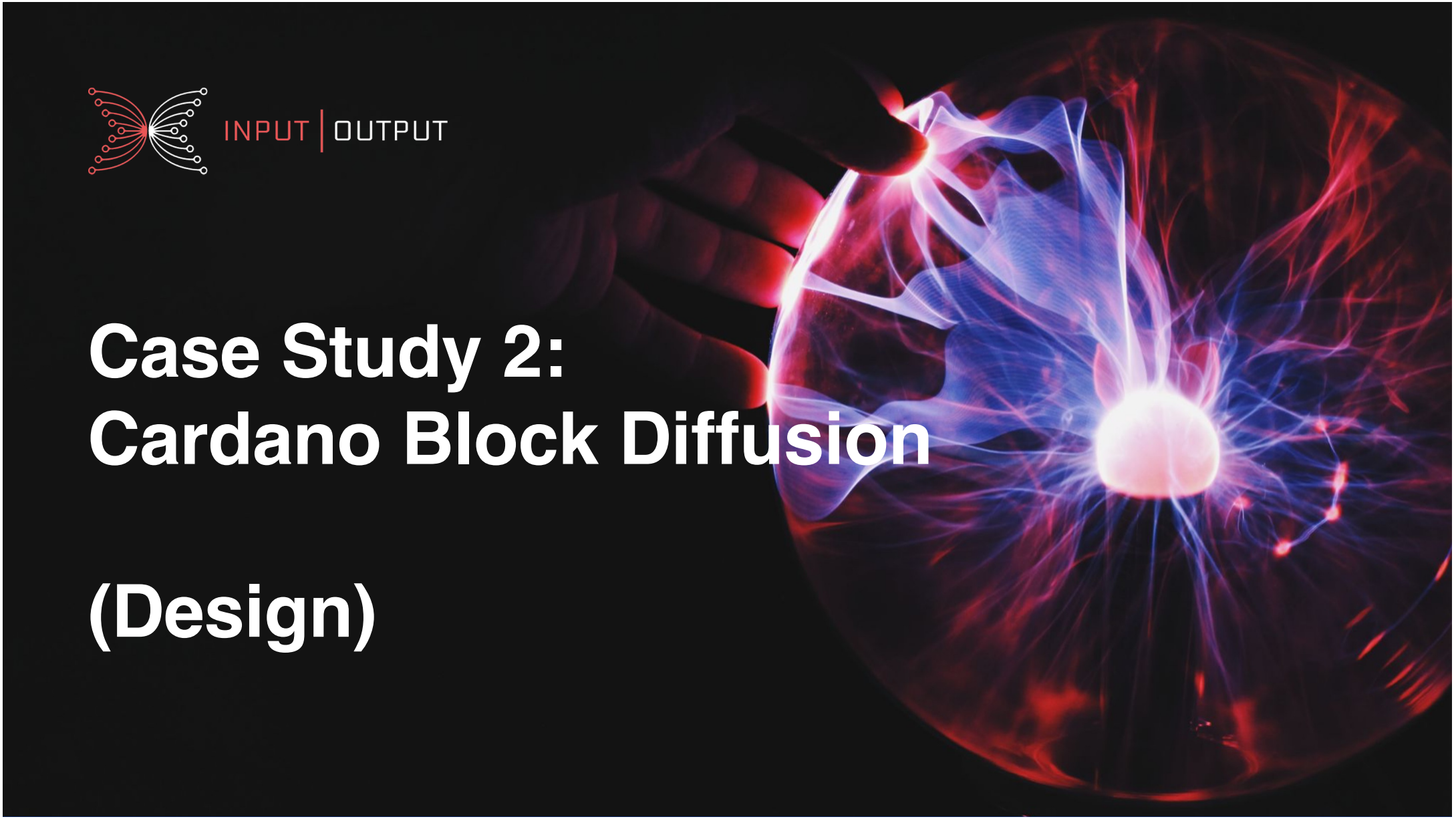
Observation points were placed at the RNC (Radio Network Controller) and around the network edges

Diagnosis approach and solution

ΔQSD was used for the diagnosis

- **Determine outcome diagram for end to end delivery of packets and measuring ΔQ for intermediate points**
- **Isolate cause and effect to pinpoint the problem, finding where loss and delay are introduced in an unexpected pattern**
- **Ultimately, to find solution**

iPhone launch findings using ΔQSD



Case Study 2: Cardano Block Diffusion

(Design)

Cardano blockchain case study

The Cardano blockchain supports the Ada cryptocurrency

- **An important part of Cardano is block diffusion, to allow an authorized node to create a block and add it to the most recently created block**
- **The initial implementation, Jormungandr (Rust), had insufficient performance**
- **The new Shelley implementation (Haskell) used ΔQSD to guide the design**
  - **From the start**
  - **Achieved required performance in a decentralised environment**

Main Design Requirement

5s block diffusion time for 95% of the network

World-wide

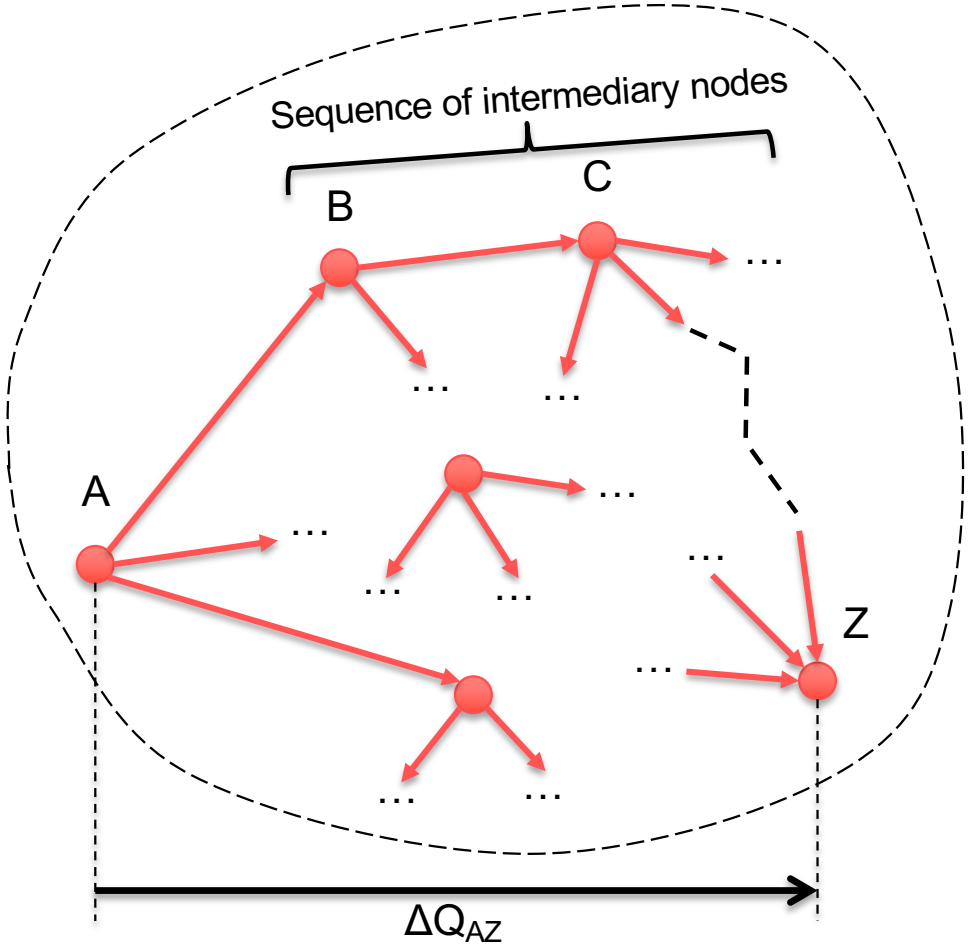
Dynamic network

Over Public TCP/IP Internet

Unreliable

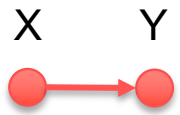
Block diffusion problem statement

Node graph of Cardano blockchain



Problem:

- Determine ΔQ_{AZ} for randomly chosen nodes A and Z, as function of design
- Determine design so that ΔQ_{AZ} satisfies performance constraints
- ΔQ_{XY} is known (measured)



Design parameters:

- Frequency of block production
- Node connection graph
- Block size
- Block forwarding protocol
- Block processing time

INPUT | OUTPUT

Step 1:

Measuring ΔQ

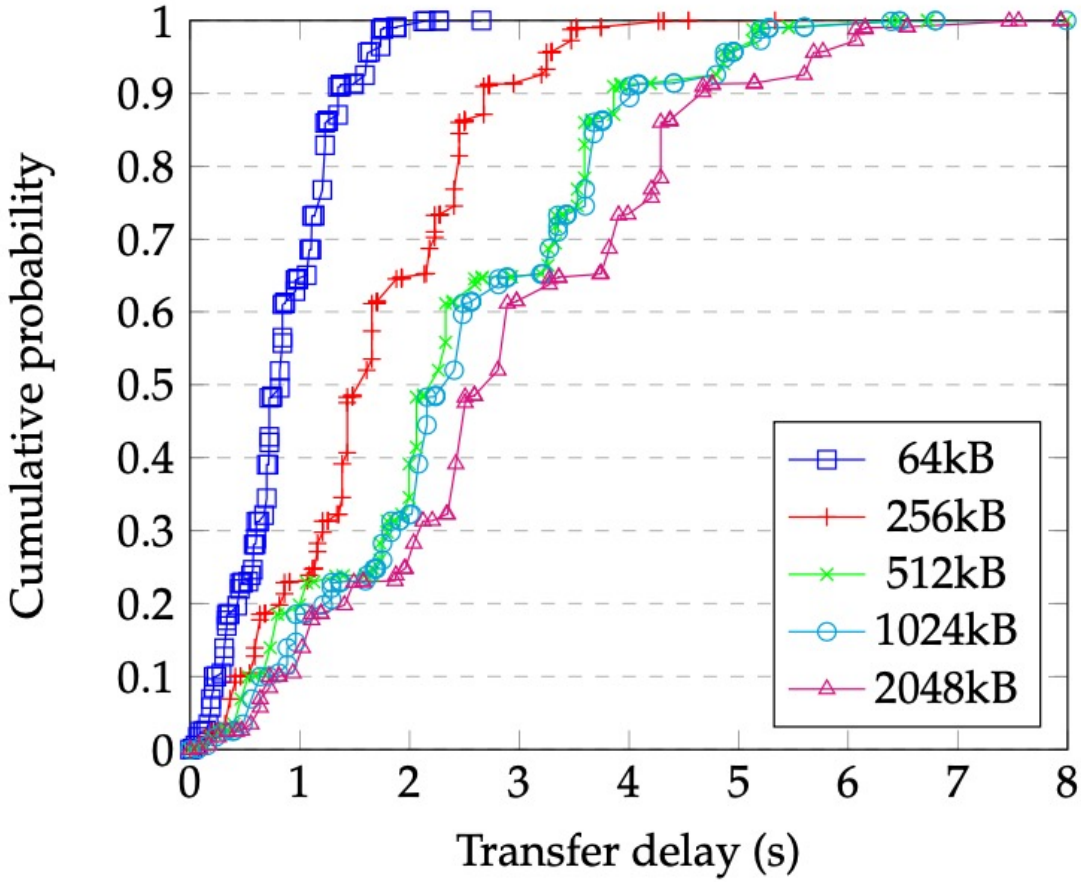
Measuring ΔQ

First step is to measure ΔQ between two Internet nodes

Four main factors

- **Block size: 64KB to 2048KB (5 steps)**
- **Network speed: measured TCP speeds**
- **Geographical distance (for single packet):**
  - **Short (same data centre), medium (same continent), long (different continents)**
- **Network congestion: initially ignored**

Measured ΔQ for varying paths

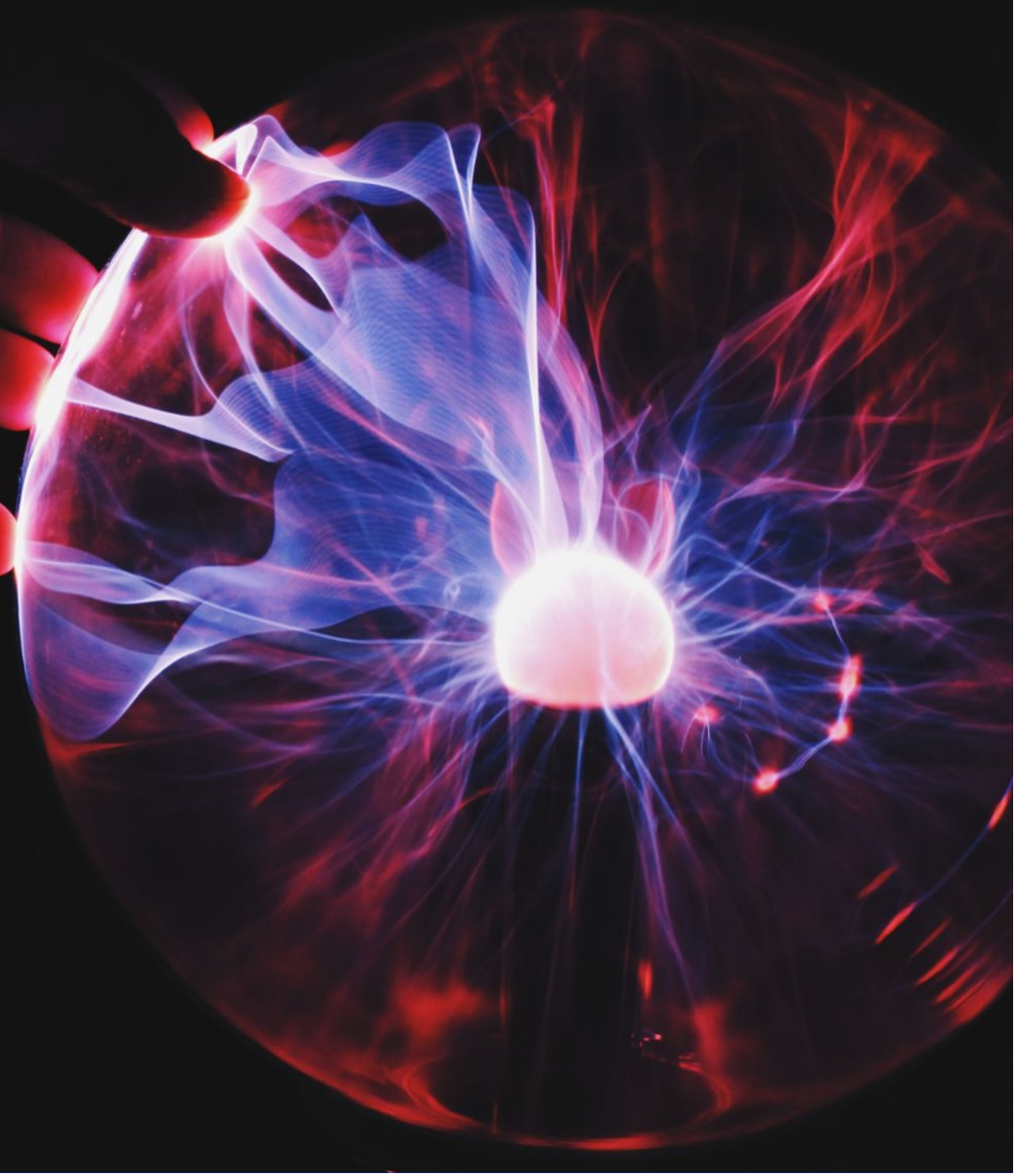


ΔQ computed for varying path lengths

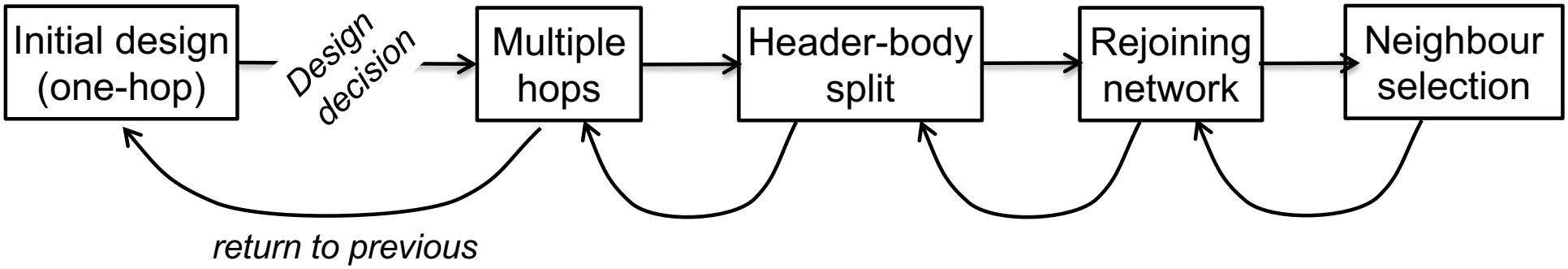
- Percentage of paths of given length in a random graph of 2500 nodes of degree 10
- Computed using probabilistic choice operator

INPUT | OUTPUT

Step 2: Designing with Outcome Diagrams



Block diffusion design using ΔQSD

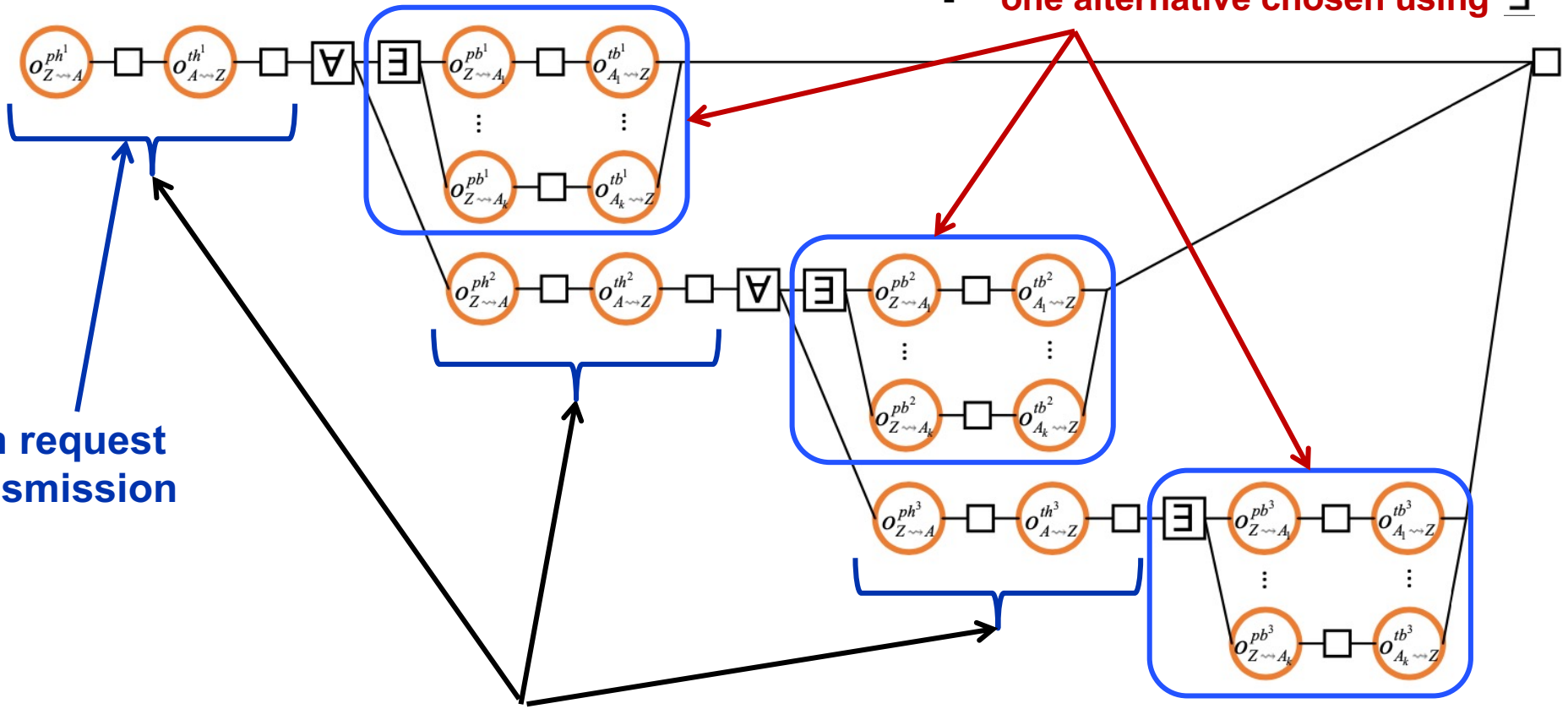


Second step: design the algorithm

- **Make design decisions and refine the outcome diagram to take each decision into account**
- **Each refinement defines a new outcome diagram and computes its ΔQ**
  - **At each refinement step, we decide whether to**
    - **keep the design; or**
    - **go back to a previous design and make another design decision**

"Mind Your Outcomes", Computers 2022, 11, 45
https://www.mdpi.com/2073-431X/11/3/45

Obtain one block body
- **from fastest neighbour**
- **one alternative chosen using ∃**



Permission request before transmission authorized

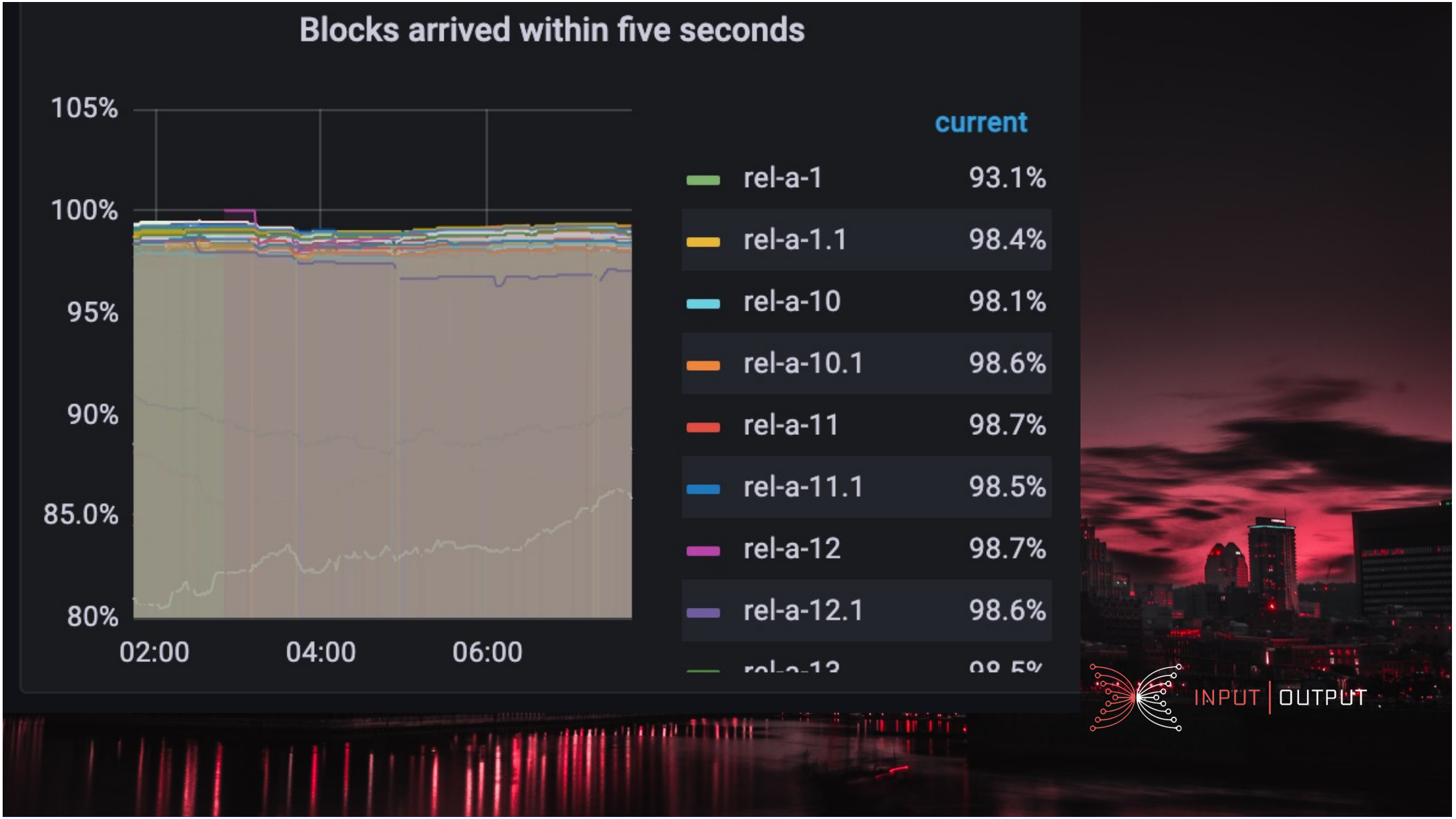
Obtain three blocks in order:
- header obtained before body
- body and next block combined using ∀

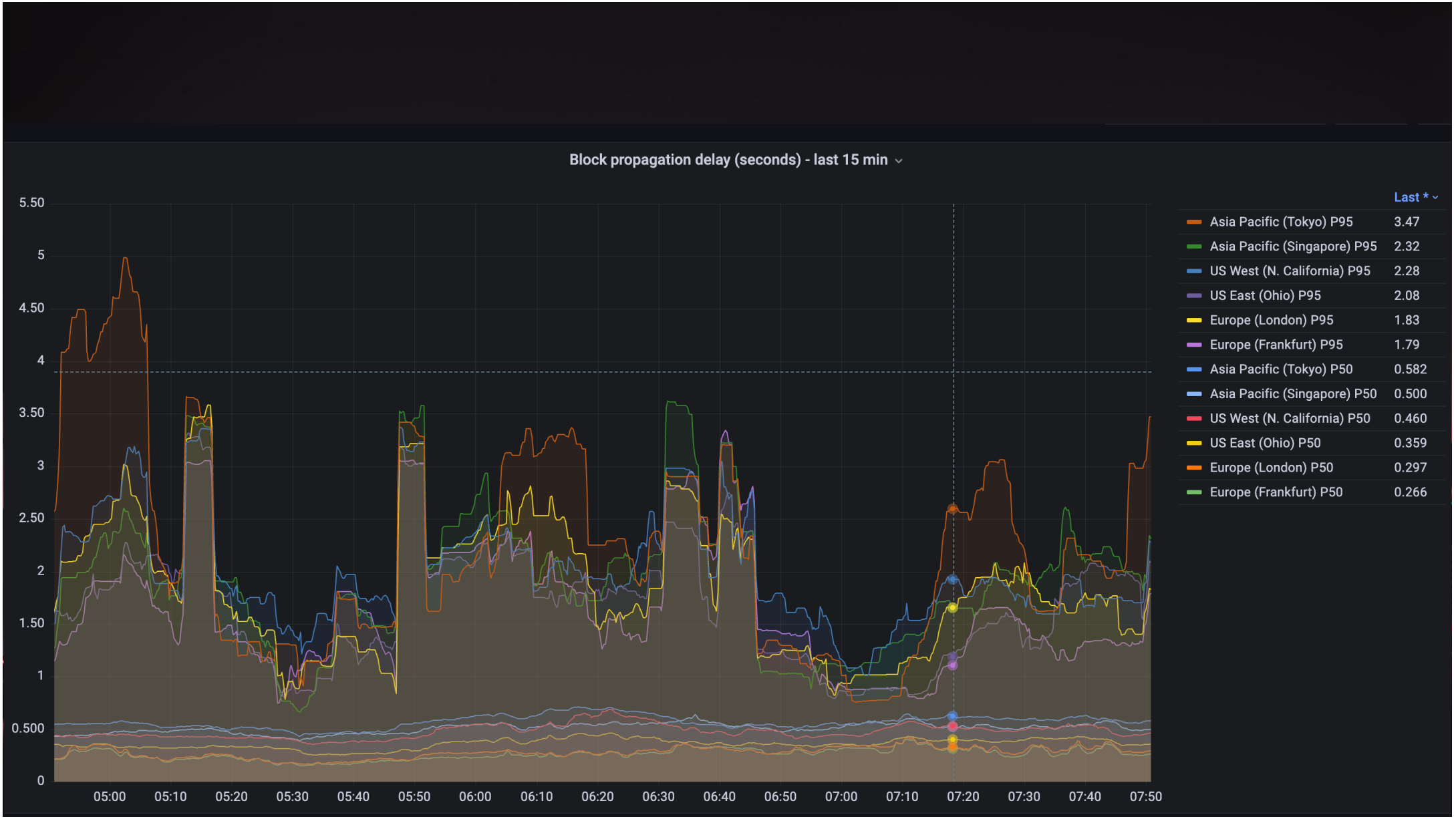
Final Result

World wide block diffusion over TCP/IP

Design ensures 95% of blocks diffused within 5s

Real-Time, Distributed System in Haskell


Blocks arrived within five seconds
105%
100%
95%
90%
85.0%
80%
02:00
04:00
06:00
current
rel-a-1 93.1%
rel-a-1.1 98.4%
rel-a-10 98.1%
rel-a-10.1 98.6%
rel-a-11 98.7%
rel-a-11.1 98.5%
rel-a-12 98.7%
rel-a-12.1 98.6%
INPUT | OUTPUT


Block propagation delay (seconds) - last 15 min
5.50
5
4.50
4
3.50
3
2.50
2
1.50
1
0.500
0
05:00
05:10
05:20
05:30
05:40
05:50
06:00
06:10
06:20
06:30
06:40
06:50
07:00
07:10
07:20
07:30
07:40
07:50
Last *
Asia Pacific (Tokyo) P95 3.47
Asia Pacific (Singapore) P95 2.32
US West (N. California) P95 2.28
US East (Ohio) P95 2.08
Europe (London) P95 1.83
Europe (Frankfurt) P95 1.79
Asia Pacific (Tokyo) P50 0.582
Asia Pacific (Singapore) P50 0.500
US West (N. California) P50 0.460
US East (Ohio) P50 0.359
Europe (London) P50 0.297
Europe (Frankfurt) P50 0.266

Conclusions

Conclusions and future steps

ΔQSD uses Functional Programming Concepts

- **Compositionality, reasoning, …**

"Mind Your Outcomes", Computers 2022, 11, 45
https://www.mdpi.com/2073-431X/11/3/45

ΔQSD works with partially specified designs

- **It can use both top-down and bottom-up approaches**
- **At any point, we can check whether the system is feasible**
  - **We can eliminate infeasible approaches early on in the design process**
- **At any point, we can predict latency and throughput under high load**
  - **It saves time and money compared to full designs or building systems**

ΔQSD cleanly factors the design into three parts

- **Compositional system made of independent parts**
- **Adding dependencies between components**
- **Adding multilevel risk management**

The Implementation is written in Haskell

Future steps

Collaboration to build/improve tools

- We are looking for people to work with us on the Haskell tooling and front-end

Ongoing project to formalize ΔQSD

- We are looking for collaborators/students/interns

Collaboration to refine the methodology

- Individuals
- Companies

Collaboration to deploy the methodology

- Software development companies
- Commercial end-users
- Government agencies

There is much more:
- Practical measurement and computation of ΔQ
- Practical experience with large systems
- Shared resources and timescales applied to large systems
- …


INPUT | OUTPUT
IOHK is Hiring!
Talk to me!!


Questions?
INPUT | OUTPUT