

N things I learned trying to do formal methods in industry

Mike Dodds - Big Spec Workshop - Oct 2024



Context: I was an academic, then I wasn't

2004 → 2017

- York / Cambridge / York - PhD, postdoc, lecturer (~ junior professor)
- Separation logic, concurrency, weak memory

2017 → now

- Galois Inc - PI / principal scientist
- Formal methods for a lot of different things: parsers, crypto(graphy), crypto(currency), protocols, cyber-physical systems ...

Context: Galois does research for \$\$\$

- A contract research shop / “R&D temp agency”
- 110 people, employee-owned
- Focus on security / reliability tech (PL, formal methods, static analysis)
- Clients: DARPA / DoD, some US Gov, some commercial



Context: Galois is doing interesting things

- **CN**, a unified testing and verification tool for real C code. Part of the VERSE project with Cambridge, UPenn, others
- **Daedalus**, a safe parsing language. We used it to build a reference PDF parser for the PDF association
- **c2rust**, a transpiler from C to Rust which is used for several popular Rust crates. Currently working on a proposal for **DARPA TRACTOR** btw.
- **Verified cryptography** with our tool SAW/Cryptol. We've worked with AWS (amongst others) including verifying core bits of the AWS-LibCrypto library

Context: Galois is doing interesting things

- **CN**, a unified testing and verification tool for real C code. Part of the VERSE project with Cambridge, UPenn, others
- **Daedalus**, a safe parsing language. We used it to build a reference PDF parser for the PDF association
- **c2rust**, a transpiler from C to Rust which is used for several popular Rust crates. Currently working on a proposal for **DARPA TRACTOR** btw.
- **Verified cryptography** with our tool SAW/Cryptol. We've worked with AWS (amongst others) including verifying core bits of the AWS-LibCrypto library

Today I won't talk about
any of these projects

Okay, then what?

I've done a lot of formal methods 'technical sales'

*potential
client*

Hello I've heard of formal methods and I'm interested in doing a formal method

I've done a lot of formal methods 'technical sales'

*potential
client*

Hello I've heard of formal methods and I'm interested in doing a formal method

Excellent! Please tell me about your objectives / target system / team / budget ...

Me

I've done a lot of formal methods 'technical sales'

*potential
client*

Hello I've heard of formal methods and I'm interested in doing a formal method

Excellent! Please tell me about your objectives / target system / team / budget ...

Me



Success! *We scope a project that meets the client's needs*

I've done a lot of formal methods 'technical sales'

*potential
client*

Hello I've heard of formal methods and I'm interested in doing a formal method

Excellent! Please tell me about your objectives / target system / team / budget ...

Me



Success! *We scope a project that meets the client's needs*

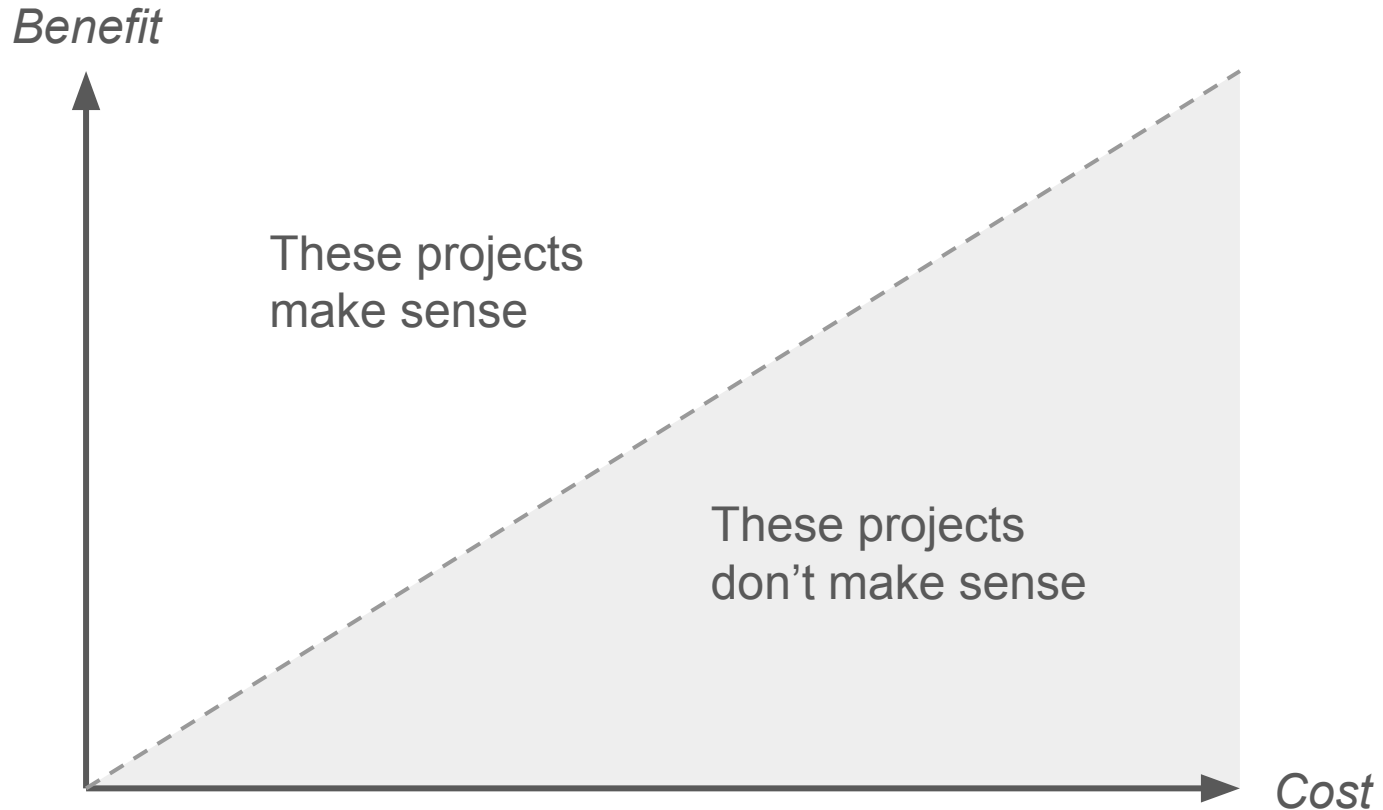


Failure! *We couldn't find a project that the client wants :(*

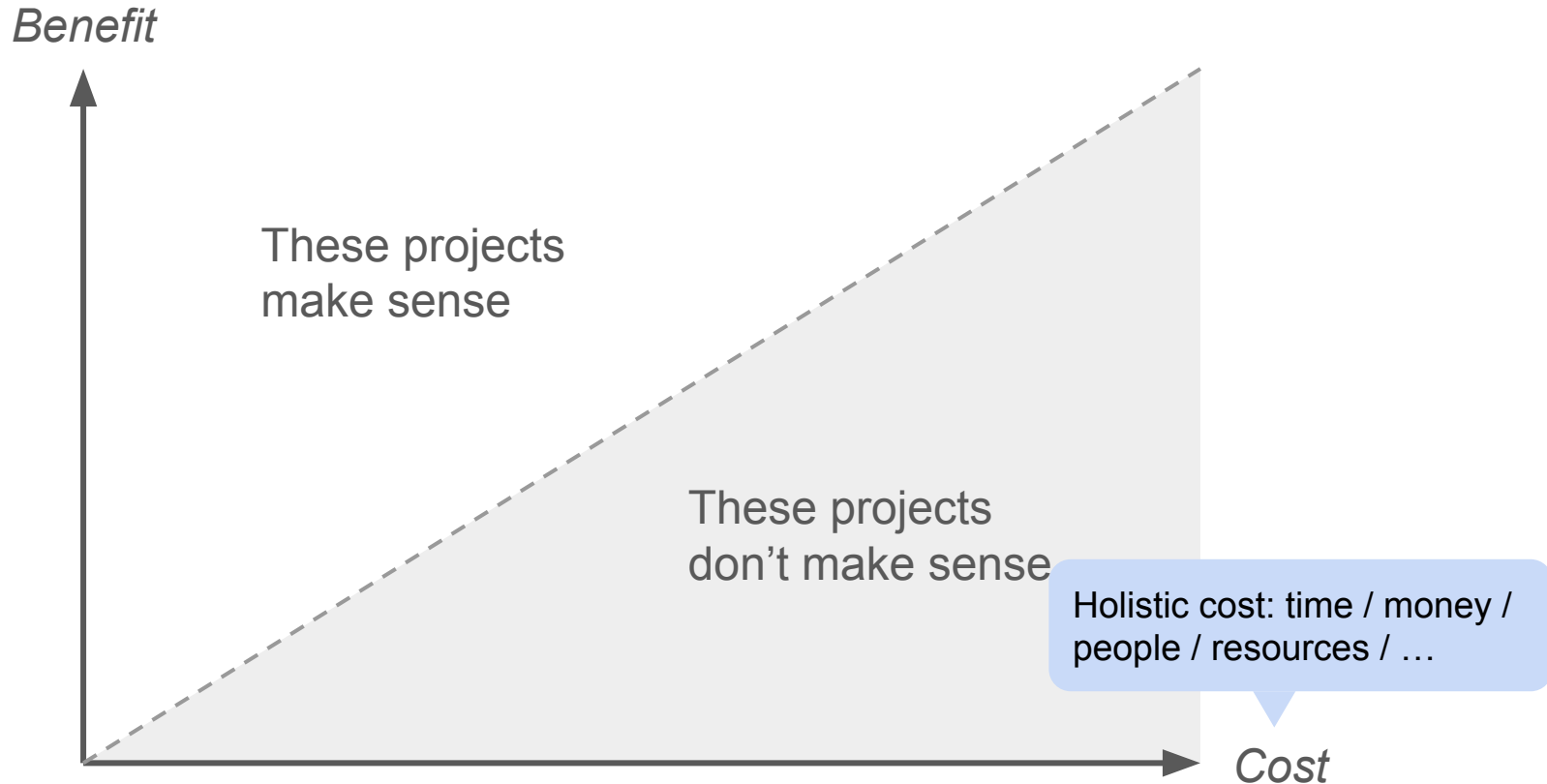
This talk:

things I learned in ~7
years of sales calls

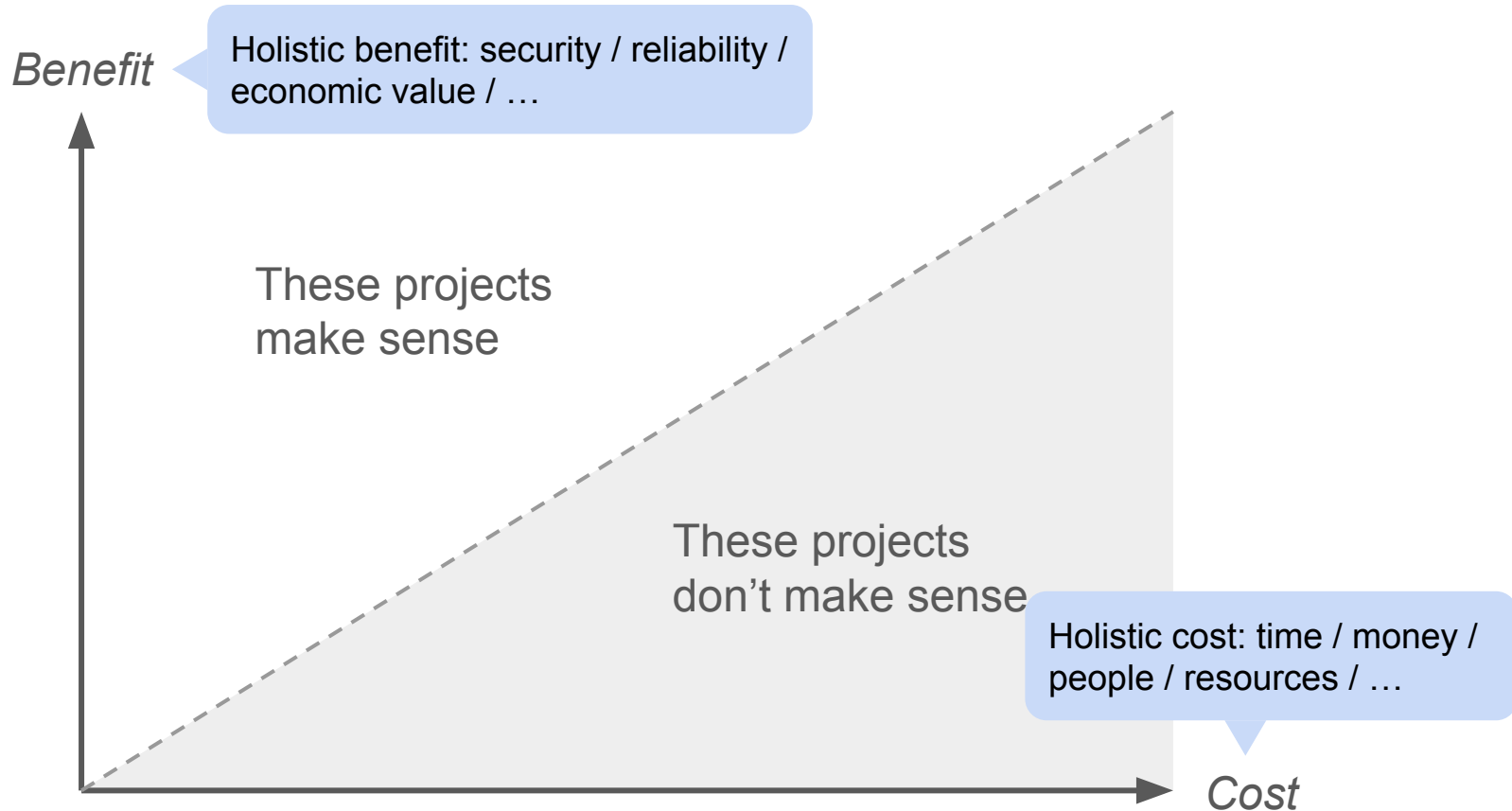
Worldview: The cost/benefit landscape of projects



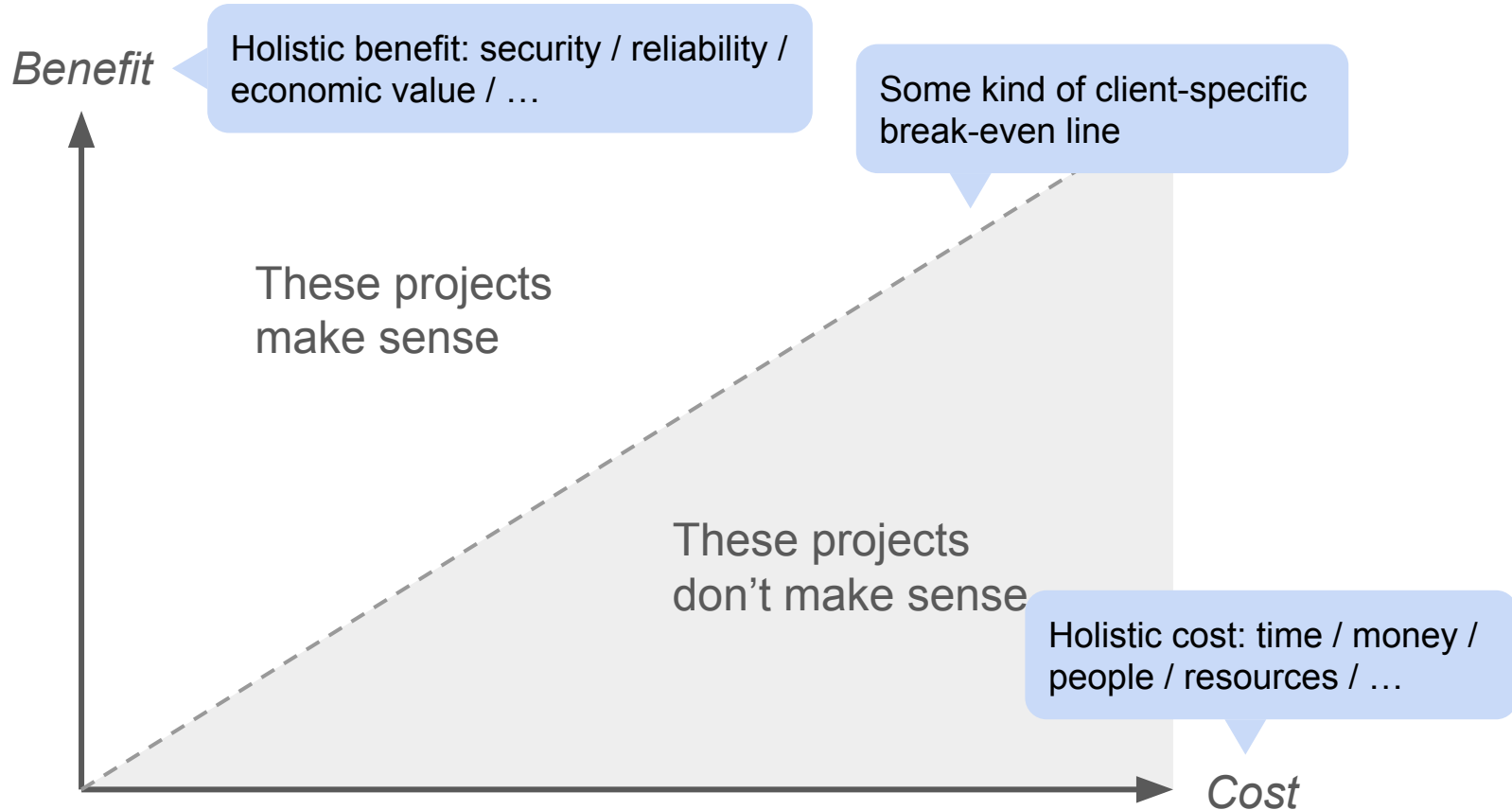
Worldview: The cost/benefit landscape of projects



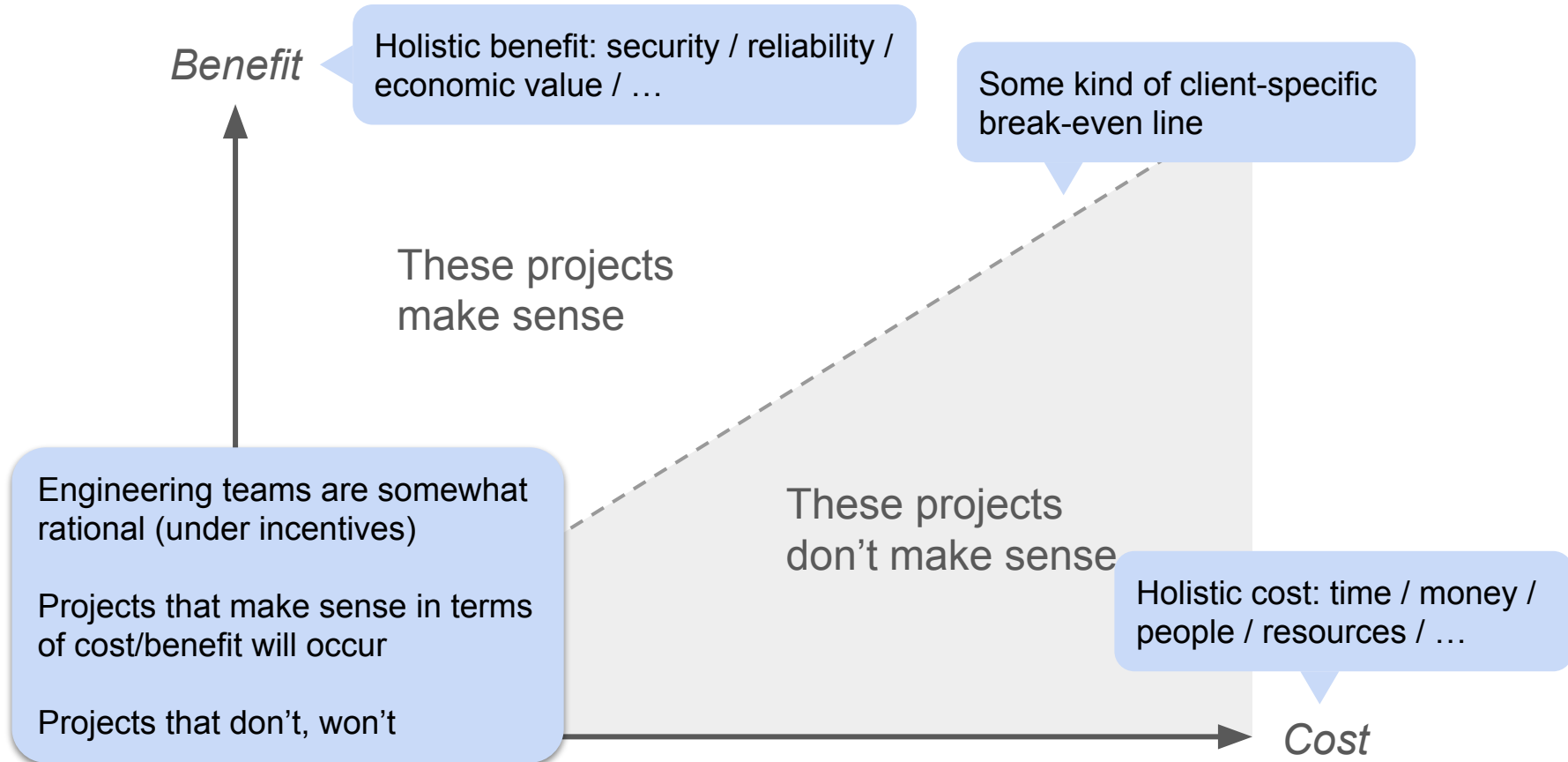
Worldview: The cost/benefit landscape of projects



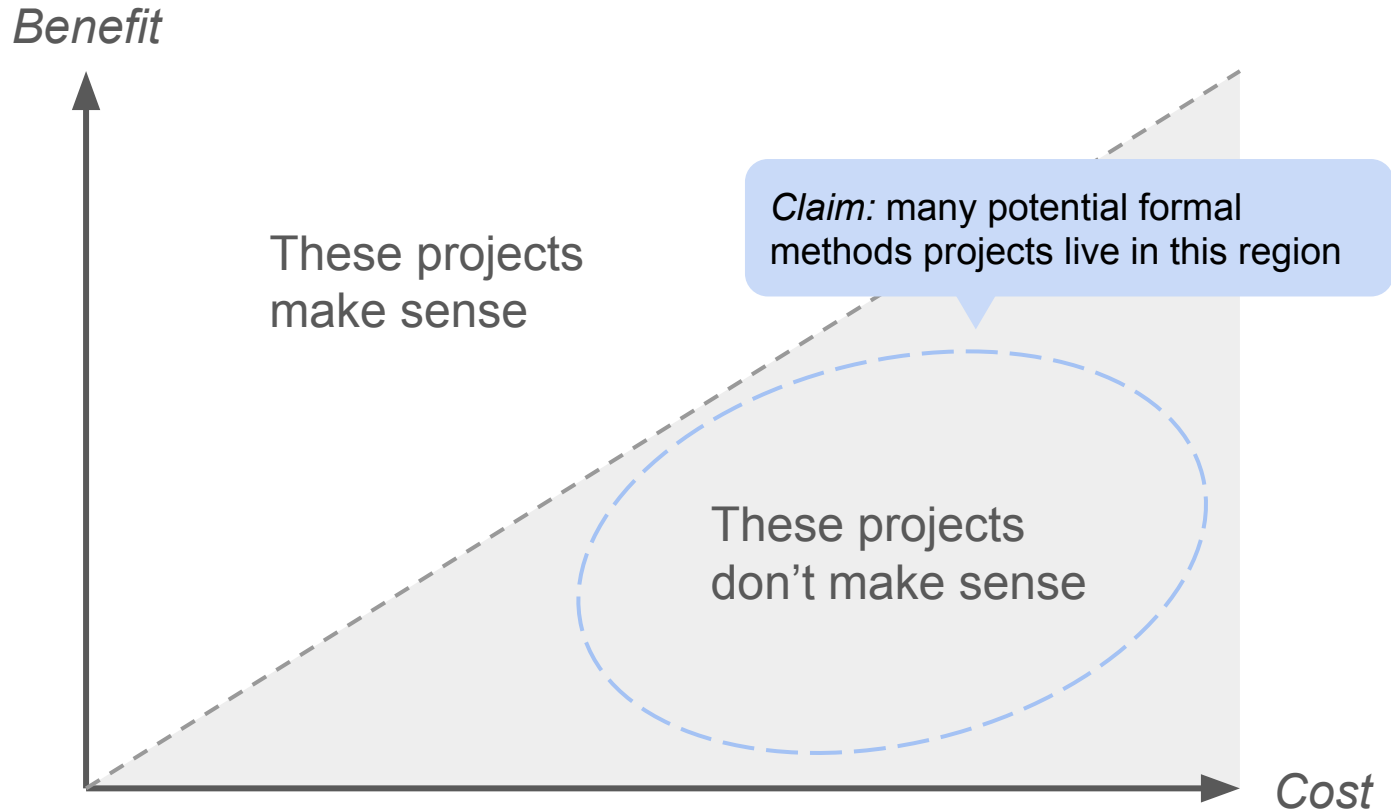
Worldview: The cost/benefit landscape of projects



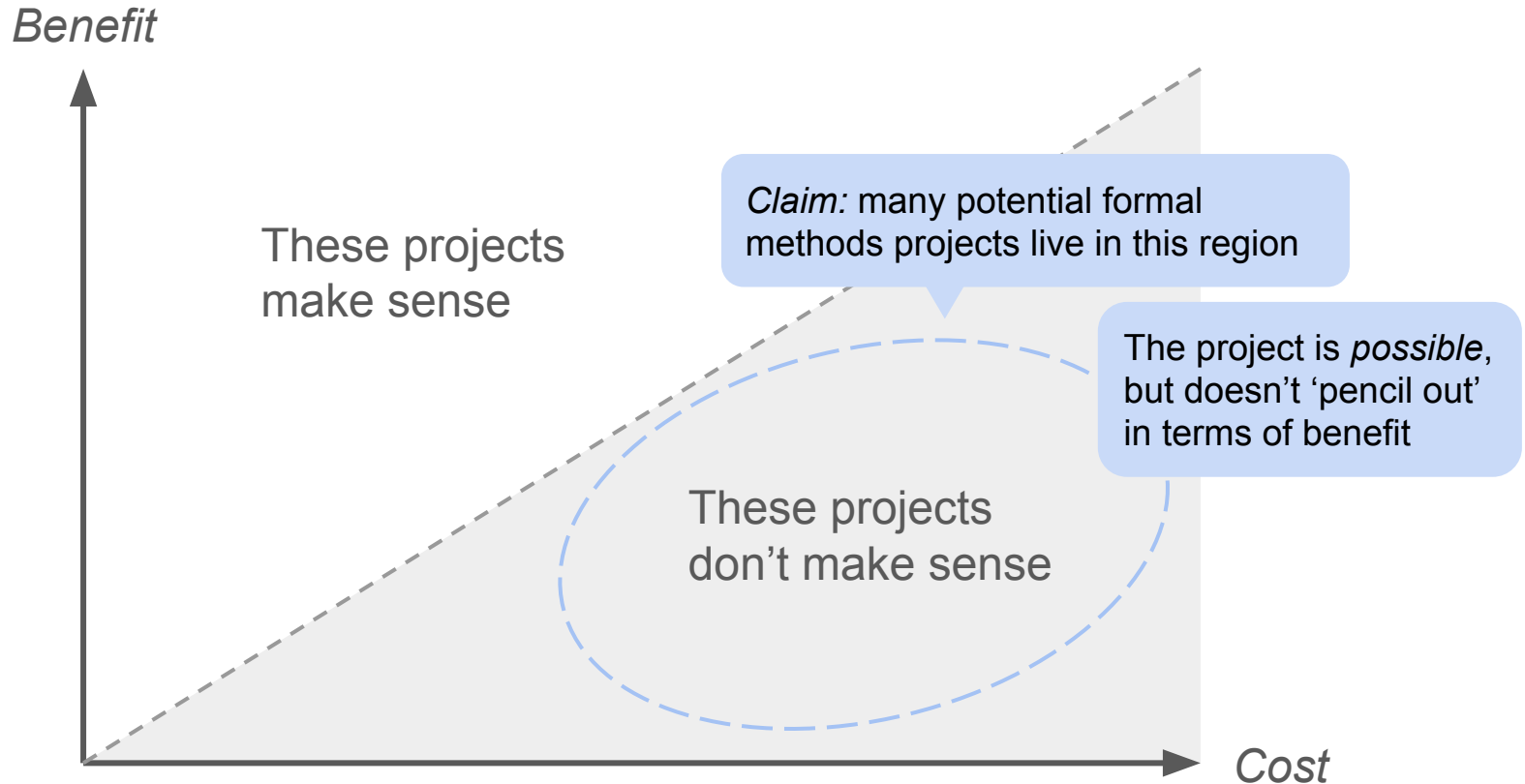
Worldview: The cost/benefit landscape of projects



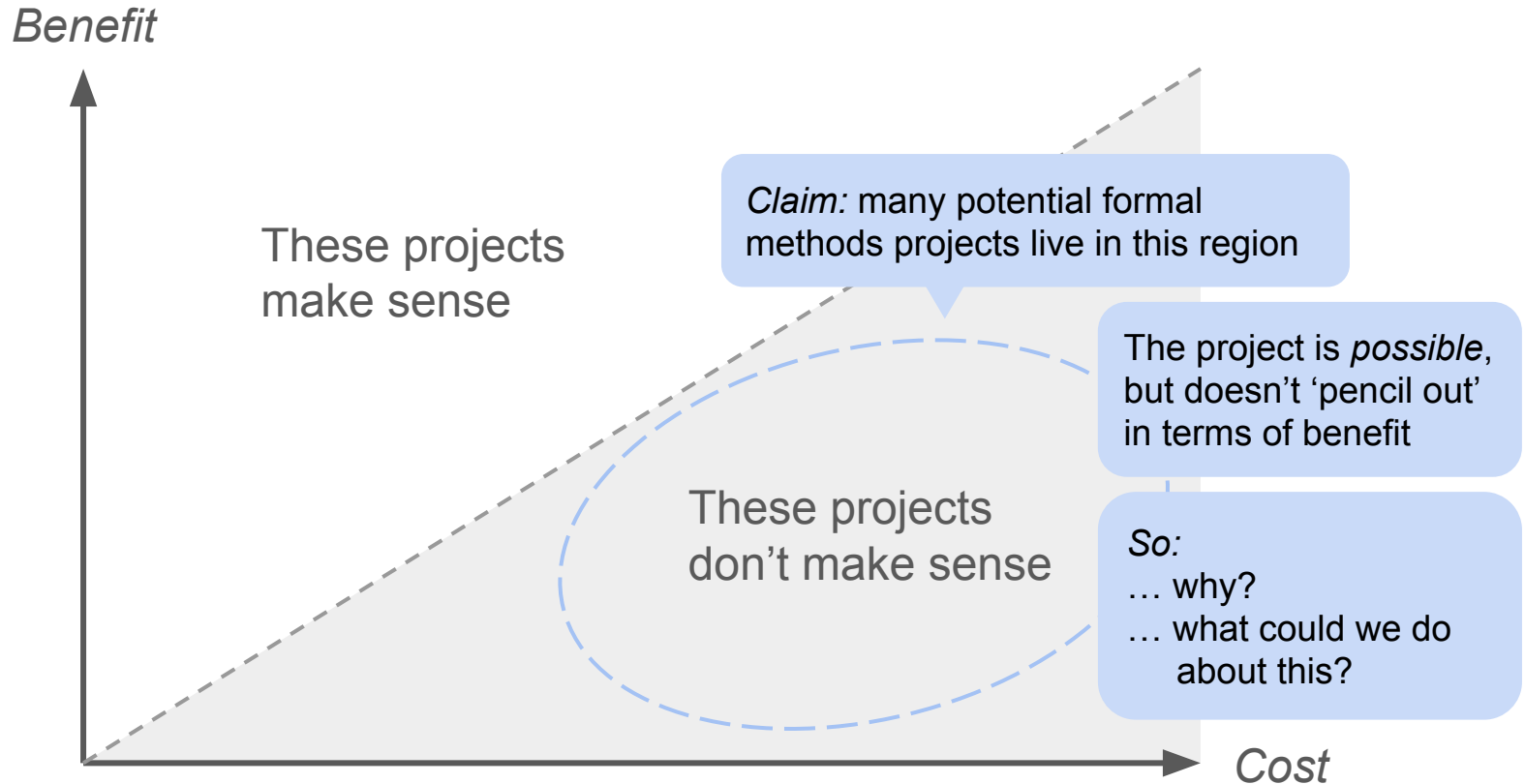
Worldview: The cost/benefit landscape of projects



Worldview: The cost/benefit landscape of projects



Worldview: The cost/benefit landscape of projects



Hot takes:

- Project have to deliver value early
- Correctness often doesn't matter
- Specifications don't exist
- It's hard to define & explain success
- Do cheap things first

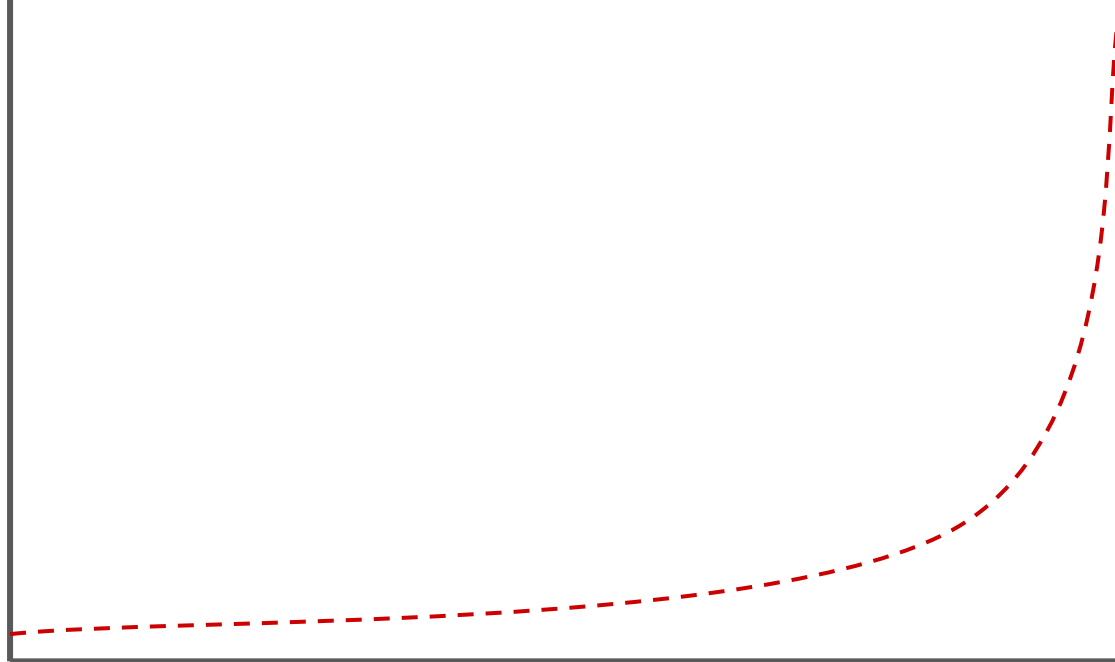
Project have to deliver
value early

A bad cost/benefit curve

Benefit



Cost (time, \$, ...)



A bad cost/benefit curve

Benefit



*Set up initial
definitions*



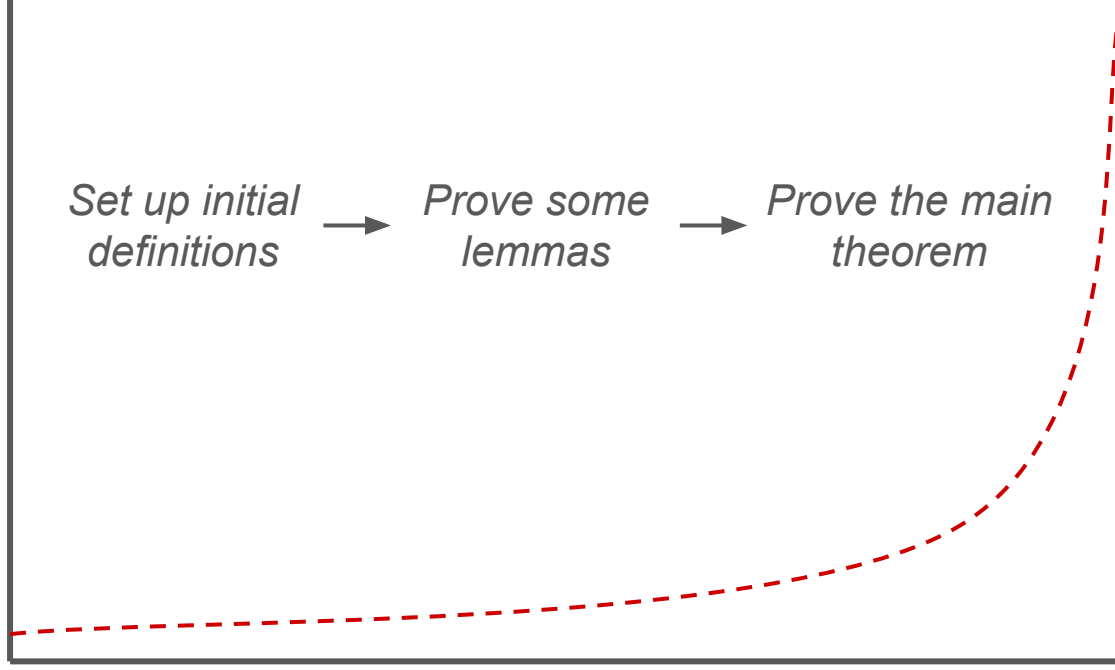
*Prove some
lemmas*



*Prove the main
theorem*



Cost (time, \$, ...)



A bad cost/benefit curve

Benefit



*Set up initial
definitions*



*Prove some
lemmas*

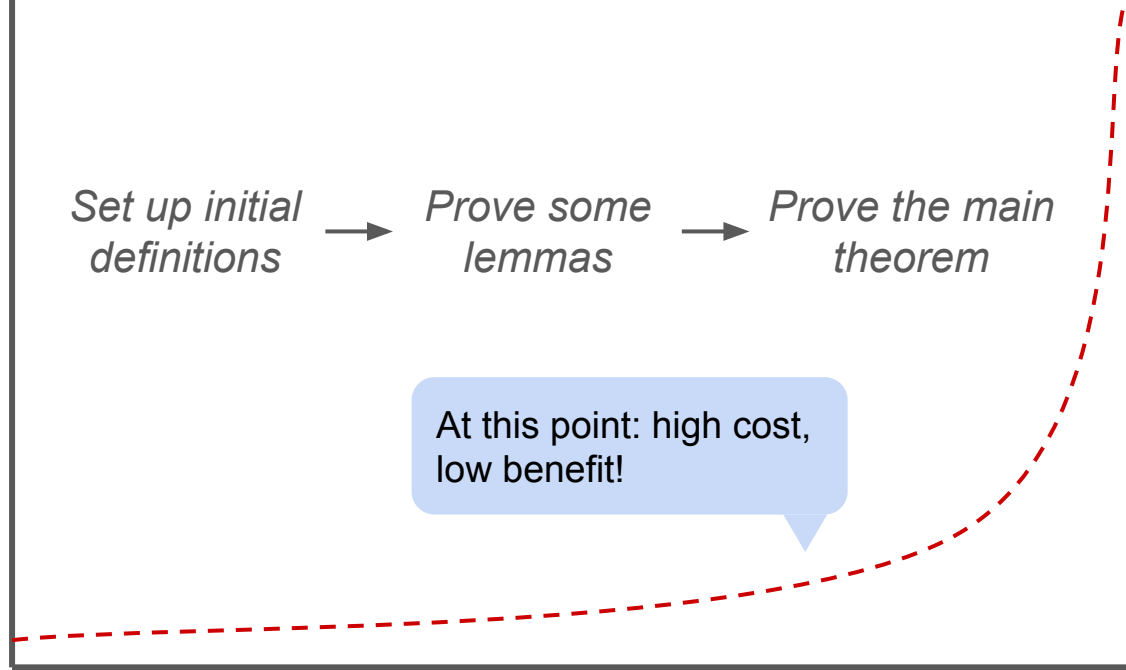


*Prove the main
theorem*

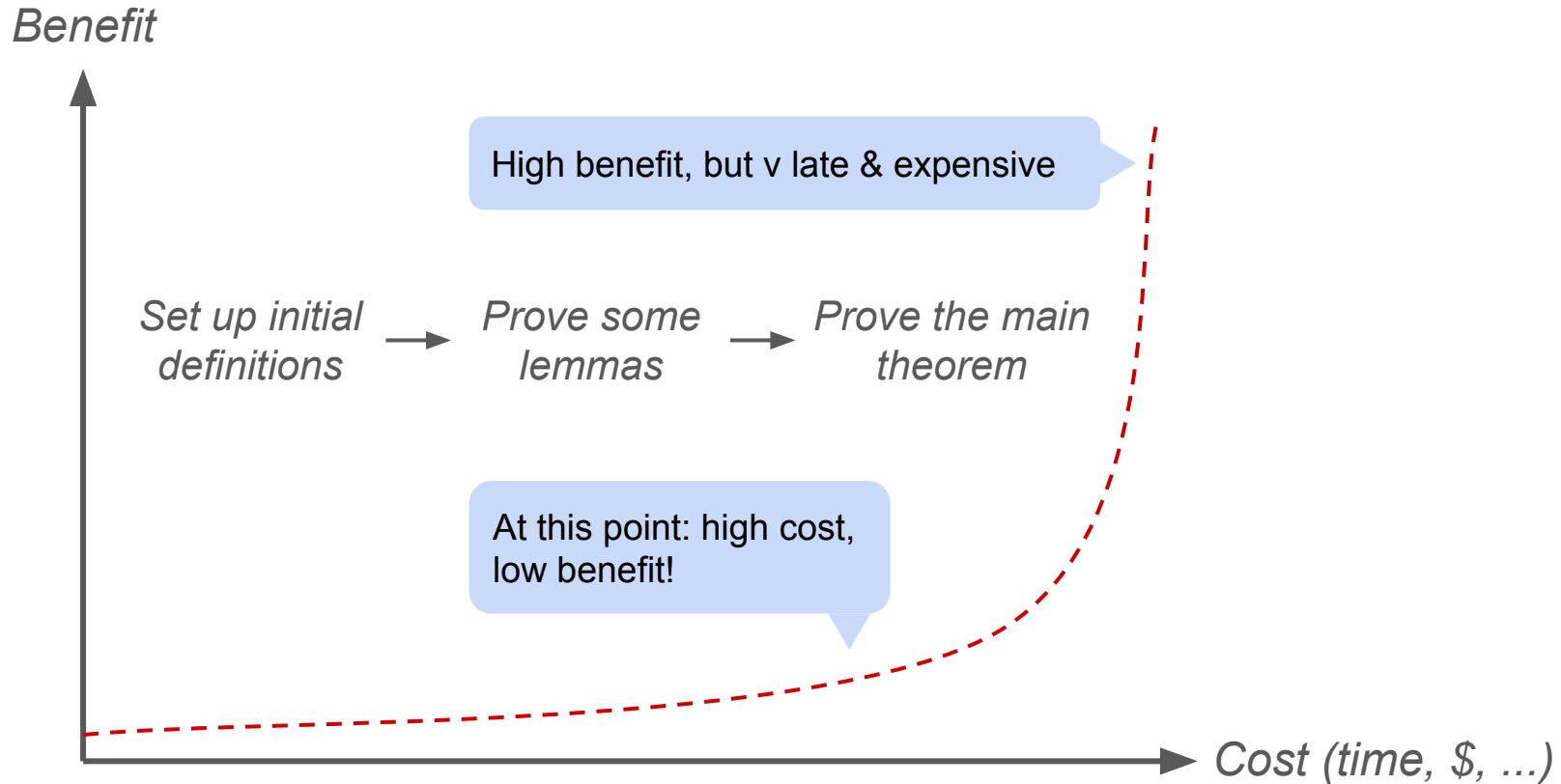
At this point: high cost,
low benefit!



Cost (time, \$, ...)



A bad cost/benefit curve

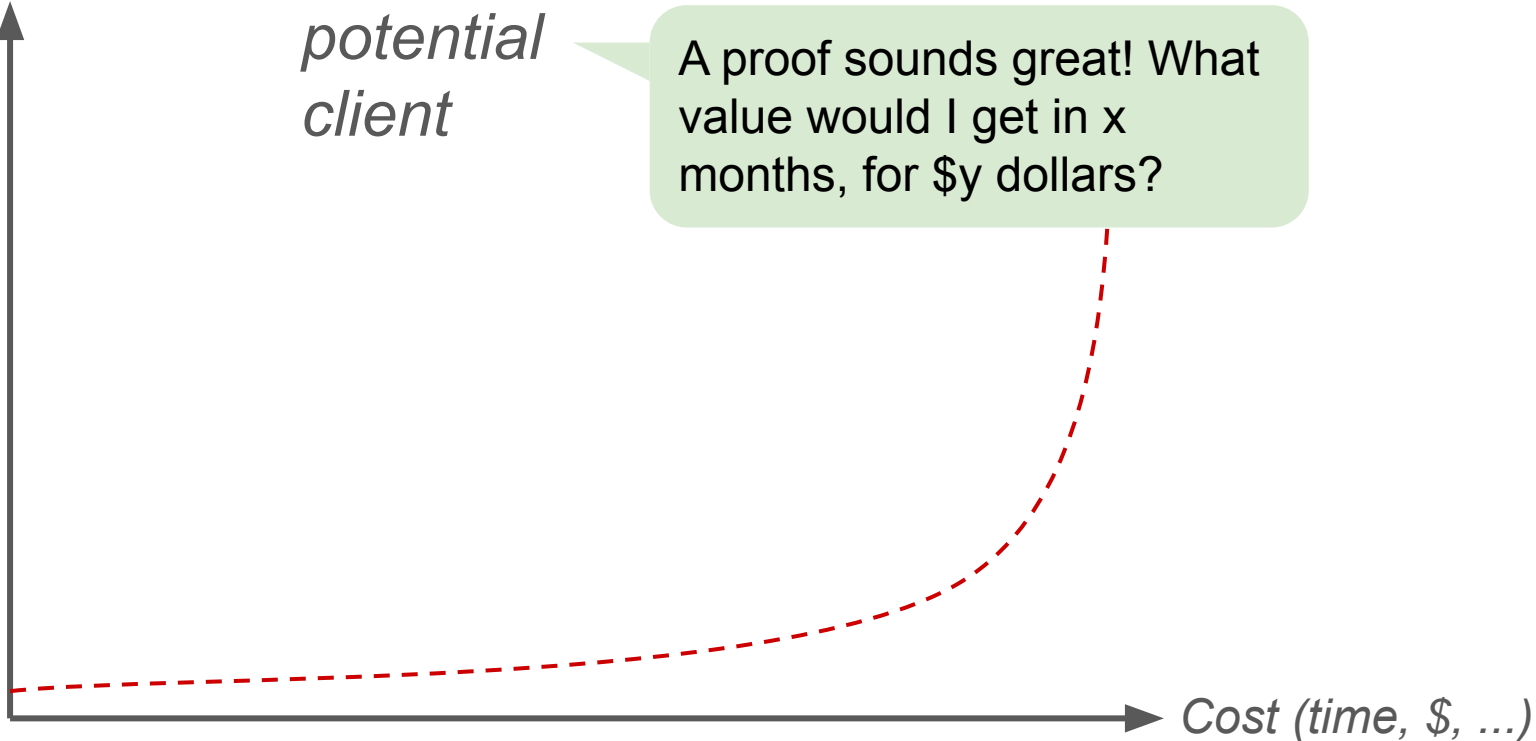


This does not work for clients

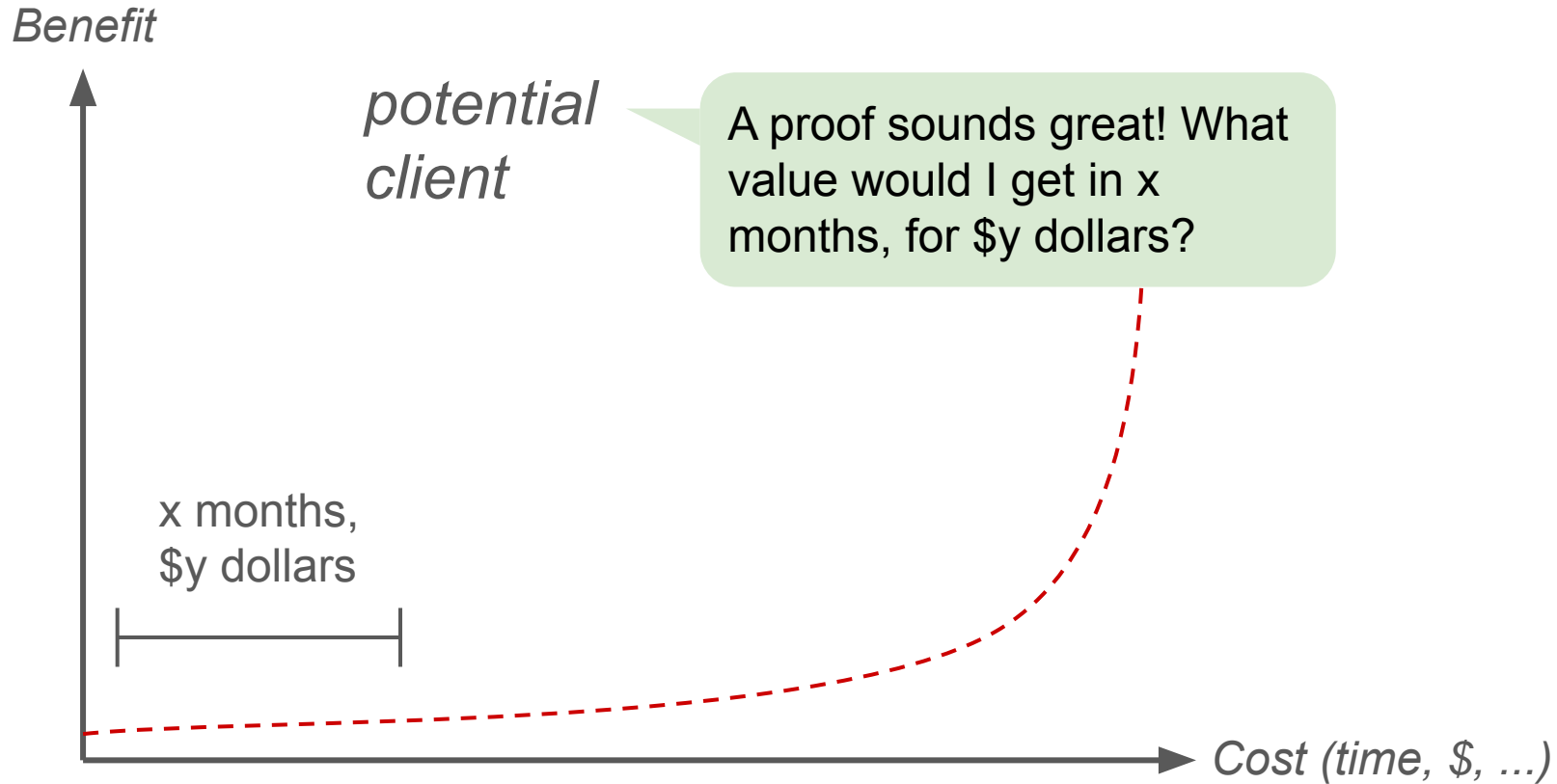
Benefit

*potential
client*

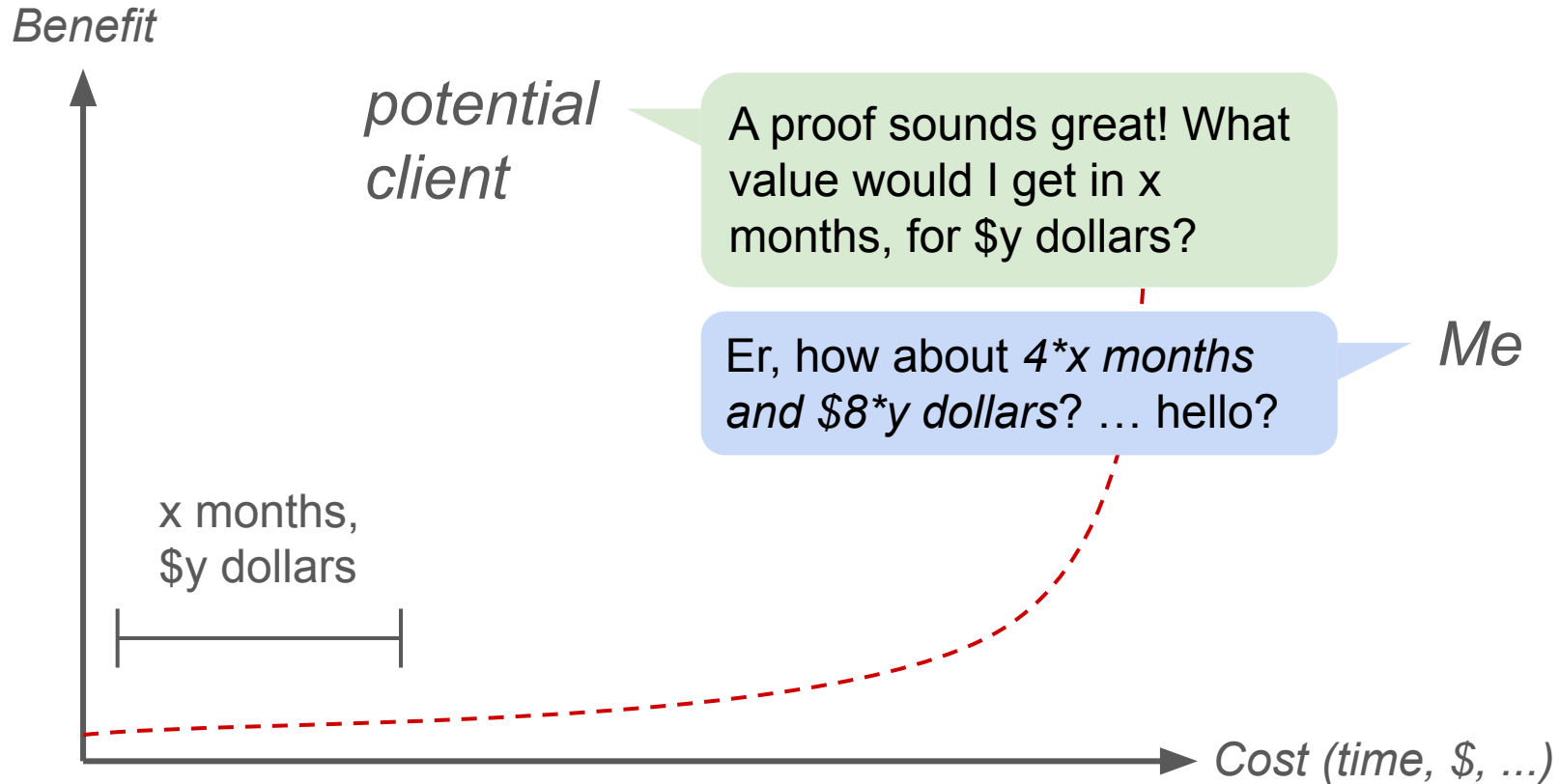
A proof sounds great! What value would I get in x months, for \$y dollars?



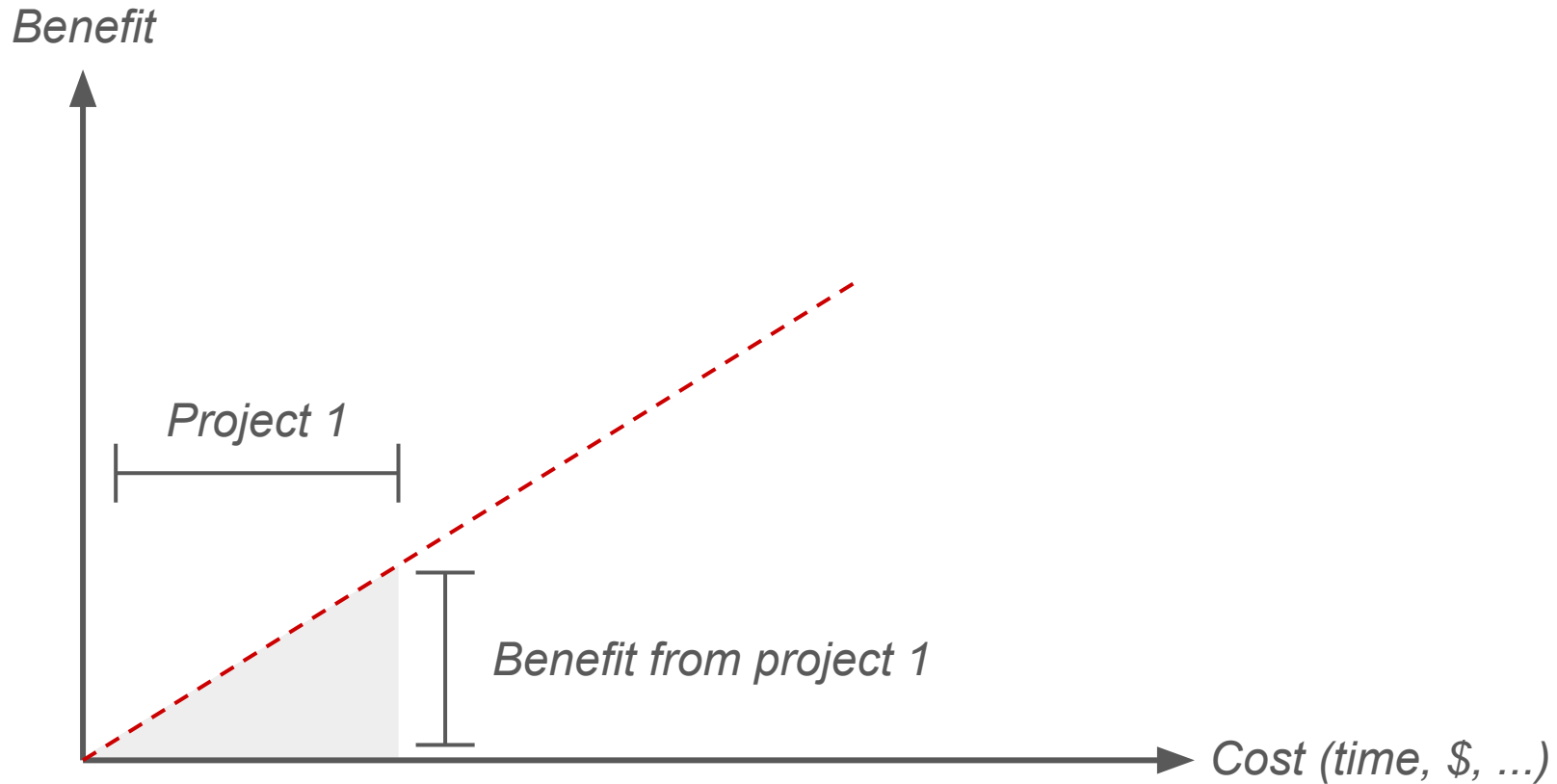
This does not work for clients



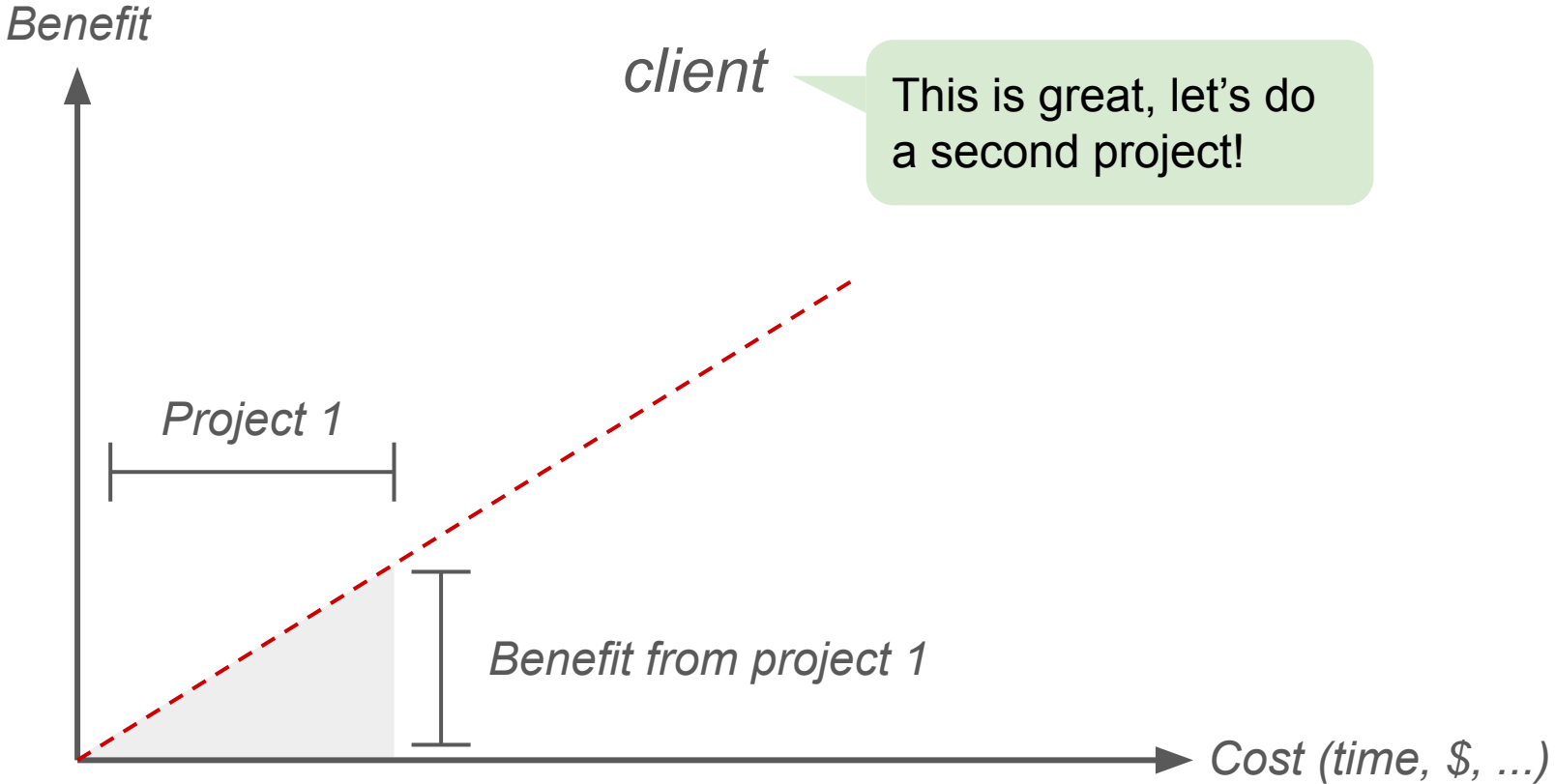
This does not work for clients



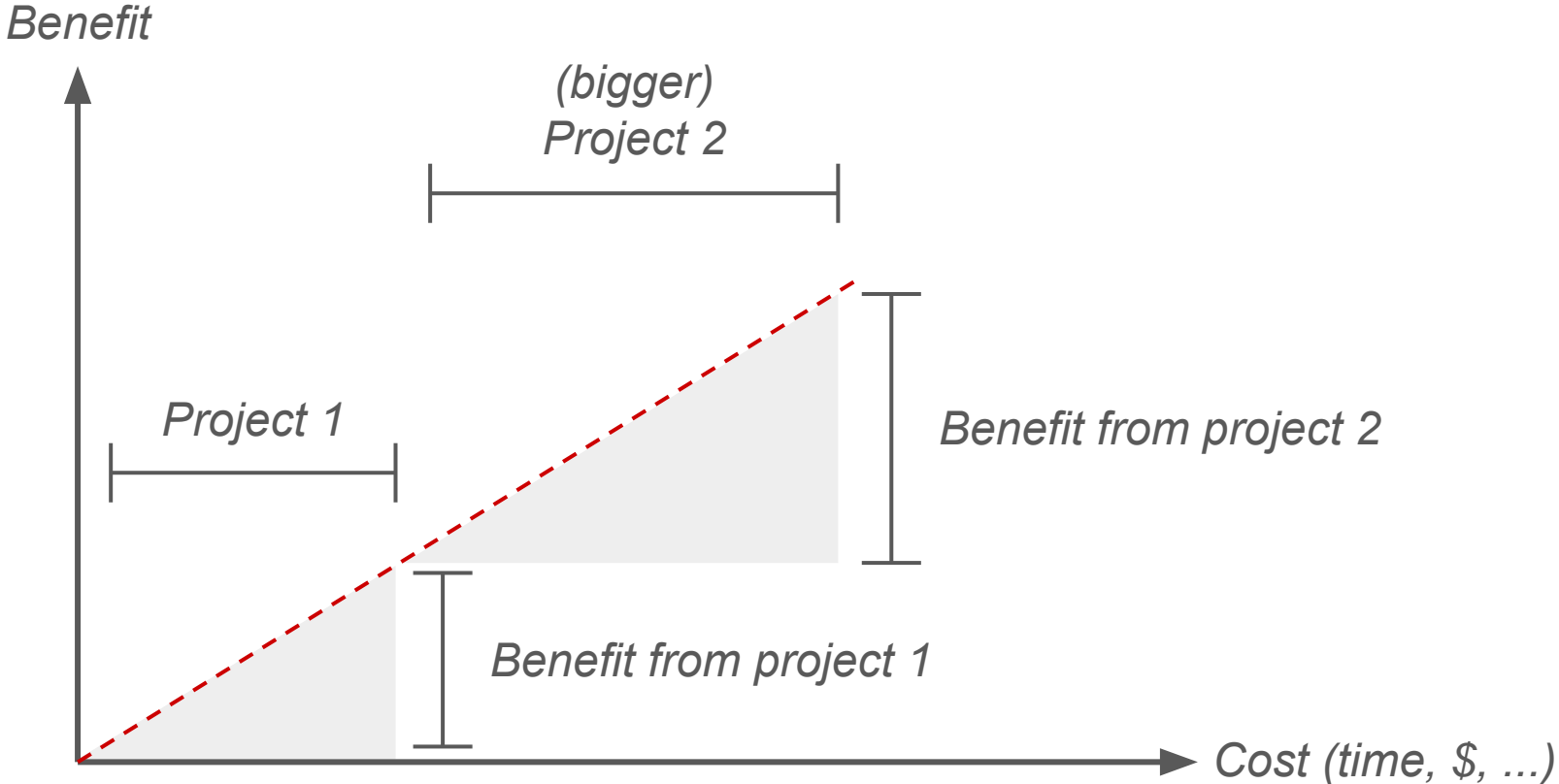
Ideally: small costs \rightarrow small but real benefits



Ideally: small costs → small but real benefits



Ideally: small costs \rightarrow small but real benefits



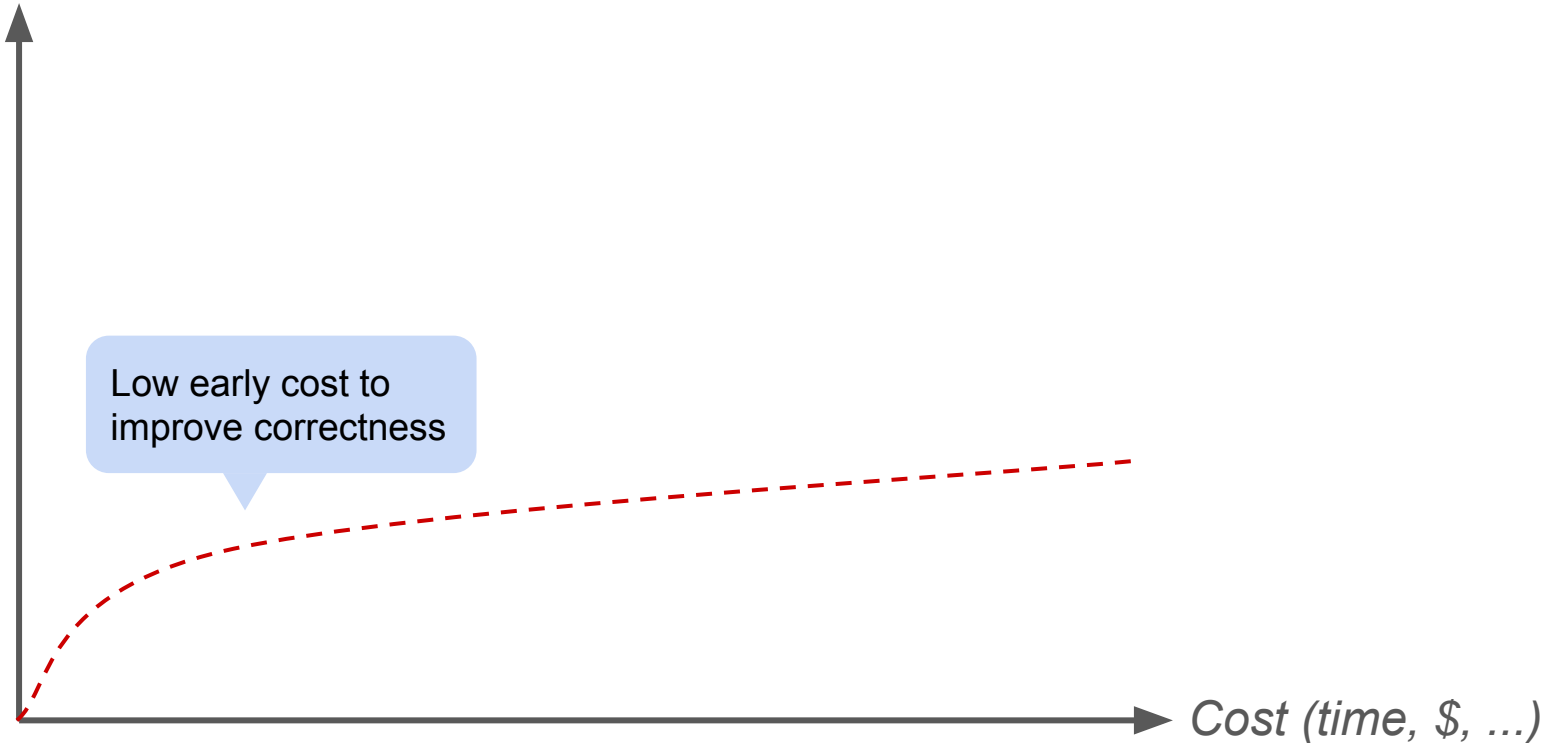
FM tools often don't work like this

Lots of up-front *costs* before we get to *benefits*:

- Writing specifications
- Building proofs
- Understanding the domain
- Training engineers
- Tool building (*sometimes*)
- ...

Compare: write-test-debug

Benefit



Low early cost to improve correctness

Compare: write-test-debug

Benefit

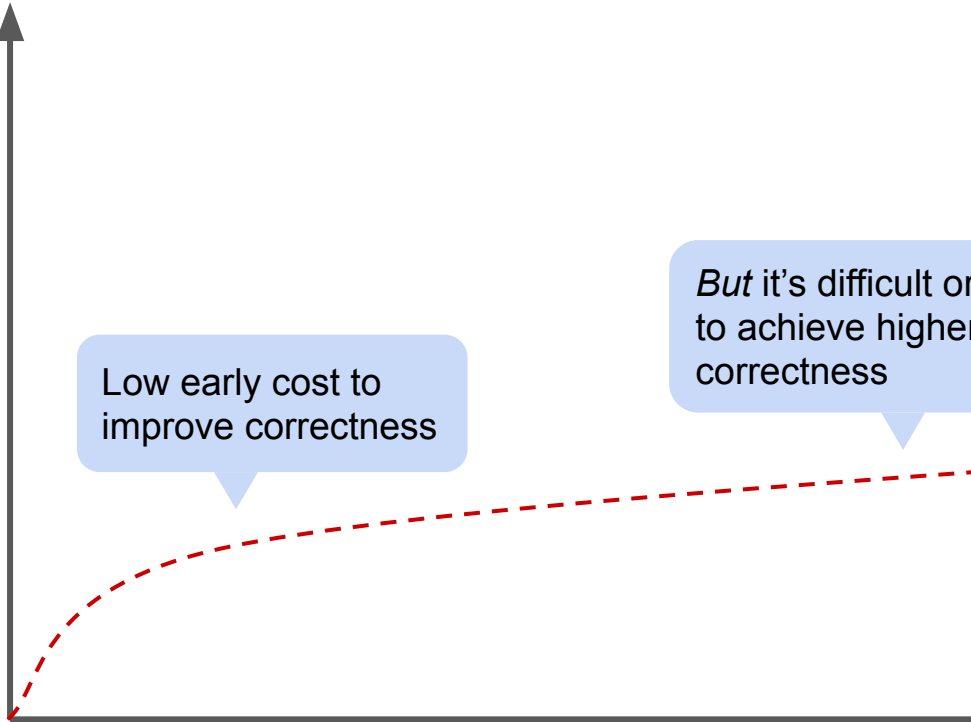


Low early cost to improve correctness

But it's difficult or impossible to achieve higher levels of correctness

(Ah-ha! Formal methods wins!)

Cost (time, \$, ...)



Correctness often
doesn't matter

Obviously at some margin correctness matters, but

- Many systems already have adequate mitigations (bugs are 'priced in')
- The marginal value of 'fewer bugs' / 'more security' can be nearly zero
- Often true for high assurance systems

Correctness might be less important than...

- Ability to ship features or security fixes more rapidly
- Ability to hire developers that can staff the team
- Paying down tech debt
- Meeting upstream needs from customers
- Reducing other costs

Especially if a correctness technology makes *any of the above more costly*

Anecdote: correctness doesn't matter

*Leadership
person at high
assurance
developer*

If we had a moderate-cost tool that would make *[system]* **2x more reliable / secure**, would that be valuable to your business?

...

*Galois
colleague*

Anecdote: correctness doesn't matter

*Leadership
person at high
assurance
developer*

If we had a moderate-cost tool that would make *[system]* **2x more reliable / secure**, would that be valuable to your business?

No

*Galois
colleague*

Anecdote: correctness doesn't matter

*Leadership
person at high
assurance
developer*

If we had a moderate-cost tool that would make *[system]* **2x more reliable / secure**, would that be valuable to your business?

*Galois
colleague*

No

Analysis:

- Their current reliability / security process works well for them
- They believe their system is already more reliable / secure than their competitors
- They don't win or lose sales on reliability / security considerations

The kicker

*Leadership
person at high
assurance
developer*

What if we could make regulatory compliance testing 10% cheaper?

*Galois
colleague*

...yes, that would save us \$Xm / year

The kicker

*Leadership
person at high
assurance
developer*

What if we could make regulatory compliance testing 10% cheaper?

*Galois
colleague*

...yes, that would save us \$Xm / year

Analysis:

- Without compliance testing, they can't sell the product
- Compliance testing is very expensive and highly manual
- Also slows down release schedule, reduces flexibility, grinding for engineers, hard to staff

Security bugs >>> other types of bugs

Non-security bugs in many environments:

- Adequately controlled by standard SWE practices
- Systems are built to resist failures, so uncorrelated bugs don't matter much

Security bugs: much more important

- An adversary can escalate a single bug into a catastrophic failure
- A lot of Galois's projects are motivated by security, rather than any other properties
- Clients typically have a threat model in mind

Useful to understand the threat model

Eg. for a cryptographic primitive, we might have:

- Memory safety violations: v bad, could lead to compromise of the host system
- Incorrect implementation of the primitive: undesirable, but less bad because it would only comprise a single message
- Timing attacks: out of scope, mitigated by some other mechanism

(I wish I saw more threat models in POPL / PLDI papers)

Specifications
don't exist

Formal specs, ideally:

Mathematically clean

Stable over time

Agreed by the users of the system

Easy to reason about

Stable, clean specs do (sort of) exist for

- Cryptographic algorithms
- Operating systems / hypervisors
- Compilers / programming languages
- Hardware

These are the big success stories for formal methods, but

- These systems are highly unusual in this regard
- Even slightly less formalizable things are very hard to deal with
- Most systems are *very un-formalizable*

Anecdote: PDF, a somewhat formalizable thing

The Portable Document Format (PDF) has:

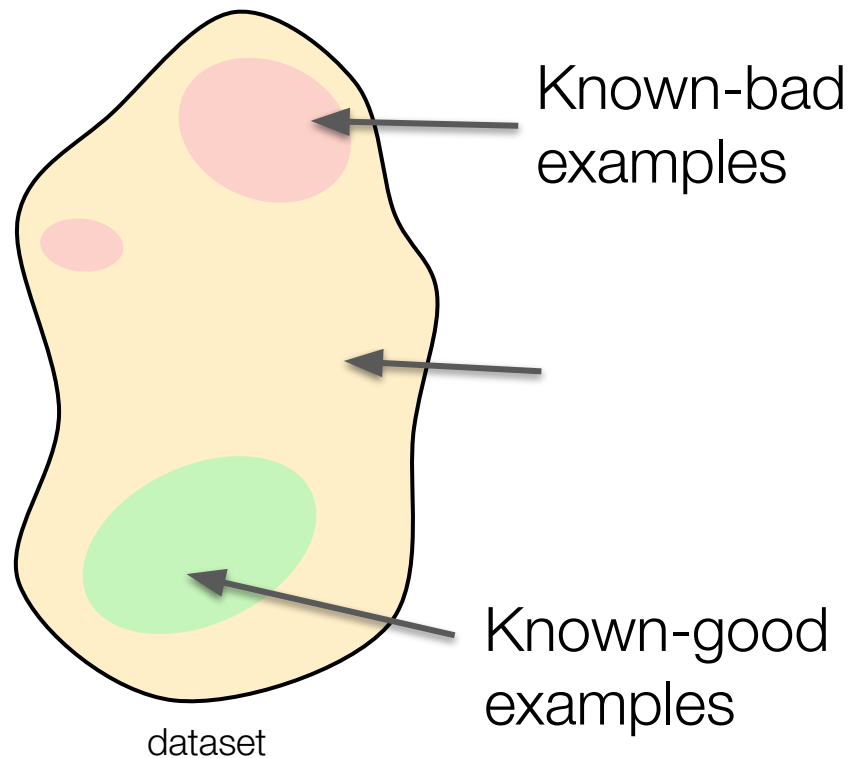
- Many parsers, many use cases
- A vigorous standards body, the PDF Association
- A large body of examples of the format

It does not have

- Agreement between existing parsers
- Specifications matching *de facto* behavior
- A clear definition of a ‘bad / insecure document’

Parsers are incentivized to parse non-conformant inputs

There's no ground truth for PDF



We used a PDF dataset of 1M+ files

Some known-good and known-bad examples, but mostly unknowable

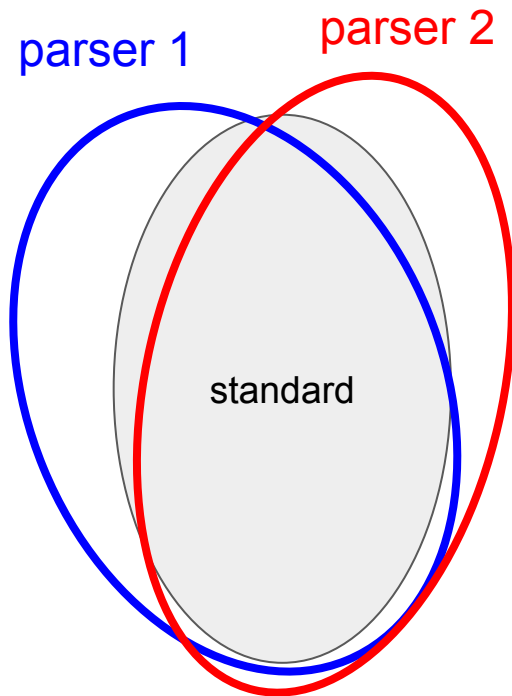
'PDF' does not exist as a coherently defined spec

We formalized PDF in our format definition language Daedalus

- Testing on millions of cases
- Worked closely with the PDF association

But...

- Non-descriptive: different from real parsers
- Non-normative: doesn't characterize bugs
- Unclear how to get to a more rigorous & accepted specification



More typical specifications

- Prose standards / RFCs / papers
- Powerpoint decks (*v common*)
- The code itself
- Reference implementations
- Inline code comments
- Test cases
- User stories
- Requirements documents
- Regulatory rules
- Scribbled notes on coffee-shop napkins
- ...

Typically these specifications are not

Mathematically clean

Stable over time

Agreed by all users of the system

Easy to reason about

We sometimes have this conversation:

Do you have a specification for [your system]?

Me

We sometimes have this conversation:

Do you have a specification for [your system]?

Me

Client

Yes, here's a 2-slide powerpoint deck

~ and / or ~

Yes, here's a 7000 page requirements document written in semi-structured prose

... which leads to this conversation:

We found that your system does X but your specification says (or implies) Y

Me

... which leads to this conversation:

We found that your system does X but your specification says (or implies) Y

Me

Client

Oh, huh, yeah that doesn't matter

~ repeat x 40,000 ~

More instances of this problem

Asking engineers to write specifications is difficult

- *Concrete*: it's hard to learn a new specification language
- *Deeper (1)*: precise specification is itself a v difficult skill to learn
- *Deeper (2)*: specification requires a more comprehensive understanding of the system than engineers typically need / have

Specifying the environment is difficult, e.g:

- Specifications for common libraries
- World modelling for a cyber-physical system

It's hard to define &
explain success

Formal methods results:

- Highly precise technical meaning (*a 600-line Lean theorem or whatever*)
- Difficult to capture as a simple explanation
- Clients probably can't read the theorem

Problems:

- How does the client know they're getting something they want?
- How do we explain our results?
- How do we know when we're done?

Explaining results (*bad version*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

Explaining results (*bad version*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

There are NO BUGS IN YOUR SYSTEM 🙌

Explaining results (*bad version*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

There are NO BUGS IN YOUR SYSTEM 🙌

~ 3 months pass ~

Client

Hey we found a horrible bug >:(

That wasn't covered by the spec / originates in some other module / [+ *technical quibbling*]

Me

Explaining results (*slightly better*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

Explaining results (*slightly better*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

To simplify, it means [*dense math theorem*]

Er...

Explaining results (*better*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

Explaining results (*better*)

We finished and proved the theorem!

Me

Client

Great, what does that mean for me?

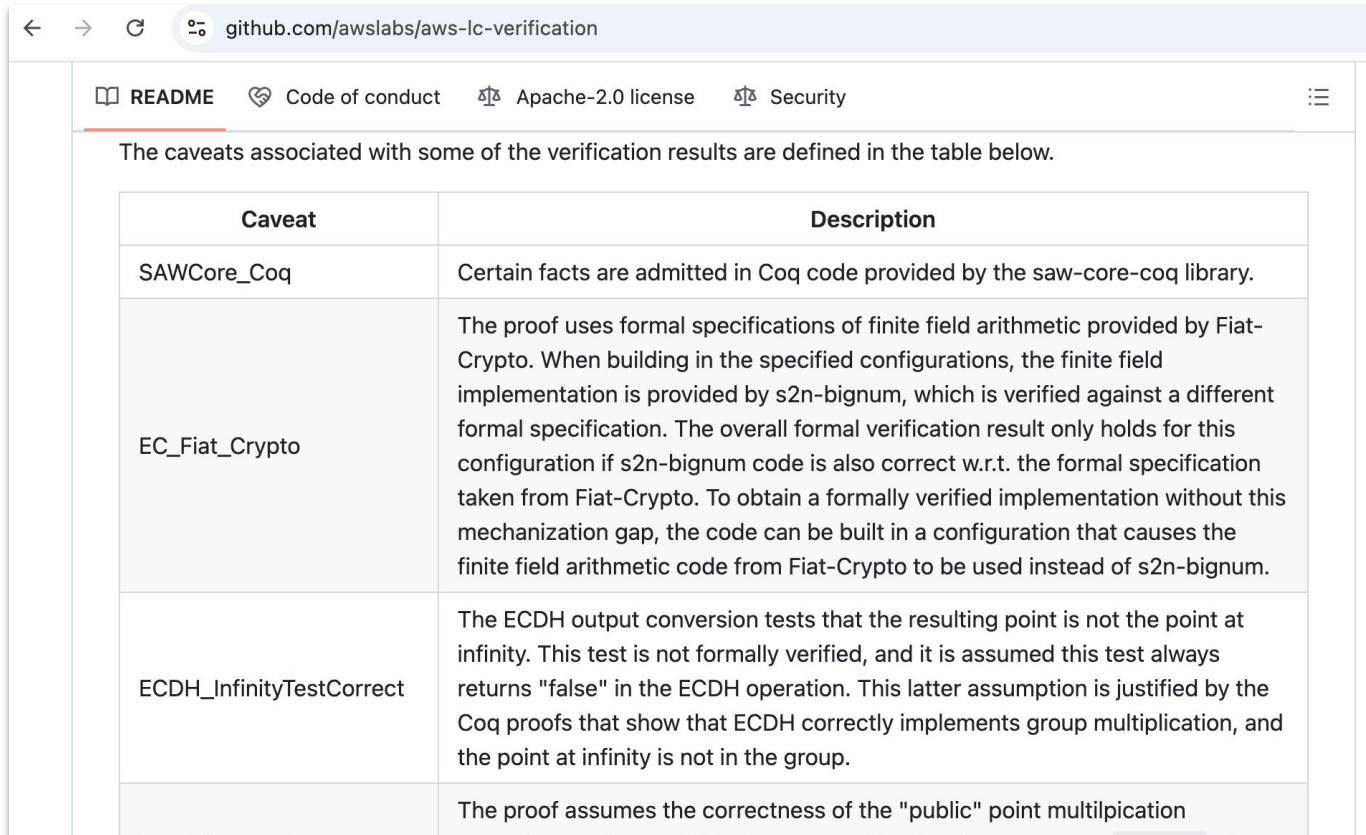
To simplify, it means [*long prose document*]

- [*caveat 1*]
- [*caveat 2*]
- ...

Does that make sense?

~ long discussion follows ~

Eg: <https://github.com/aws-labs/aws-ic-verification>



The screenshot shows a web browser displaying the GitHub repository page for `aws-labs/aws-ic-verification`. The browser's address bar shows the URL `github.com/aws-labs/aws-ic-verification`. Below the navigation bar, there are links for `README`, `Code of conduct`, `Apache-2.0 license`, and `Security`. The main content area contains a paragraph stating: "The caveats associated with some of the verification results are defined in the table below." Below this text is a table with two columns: `Caveat` and `Description`. The table lists three caveats: `SAWCore_Coq`, `EC_Fiat_Crypto`, and `ECDH_InfinityTestCorrect`. The `EC_Fiat_Crypto` and `ECDH_InfinityTestCorrect` rows are highlighted in light blue.

| Caveat | Description |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SAWCore_Coq</code> | Certain facts are admitted in Coq code provided by the <code>saw-core-coq</code> library. |
| <code>EC_Fiat_Crypto</code> | The proof uses formal specifications of finite field arithmetic provided by Fiat-Crypto. When building in the specified configurations, the finite field implementation is provided by <code>s2n-bignum</code> , which is verified against a different formal specification. The overall formal verification result only holds for this configuration if <code>s2n-bignum</code> code is also correct w.r.t. the formal specification taken from Fiat-Crypto. To obtain a formally verified implementation without this mechanization gap, the code can be built in a configuration that causes the finite field arithmetic code from Fiat-Crypto to be used instead of <code>s2n-bignum</code> . |
| <code>ECDH_InfinityTestCorrect</code> | The ECDH output conversion tests that the resulting point is not the point at infinity. This test is not formally verified, and it is assumed this test always returns "false" in the ECDH operation. This latter assumption is justified by the Coq proofs that show that ECDH correctly implements group multiplication, and the point at infinity is not in the group. |
| | The proof assumes the correctness of the "public" point multiplication |

Issues with this approach

- Expensive, depends on technical clients and a lot of discussions
- May still result in misunderstanding
- Hard for internal teams to explain to management

Also: caveats tend to get 'smoothed off'

- Galois: *[technical results, technical caveats]*
- Client engineering team: *[simplified results, simplified caveats]*
- CTO: *[simplified results]*
- PR team: "Galois has shown there are NO BUGS IN OUR SYSTEM! 🙌"

Related problem: defining completion conditions

Sometimes a caveat significantly changes project scope

- Project (with caveats 1,2,3): \$x months, \$y dollars
- Project (with caveats 1,3): \$x*3 months, \$y*5 dollars

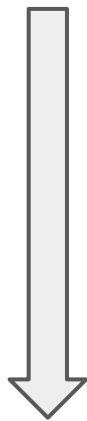
Let's hope we agreed beforehand whether caveat 2 matters

Sometimes the *landscape of caveats* is not obvious before the project starts

Do cheap things first

Cheap techniques work!

- Code review/Documentation
- Testing
- CI/CD
- Fuzzing/property based testing
- Modelling/Model Checking
- Symbolic Testing
- Program proof/Correct by Construction

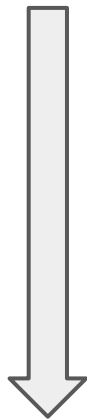


*Increasing effort,
Increasing confidence*

Cheap techniques work!

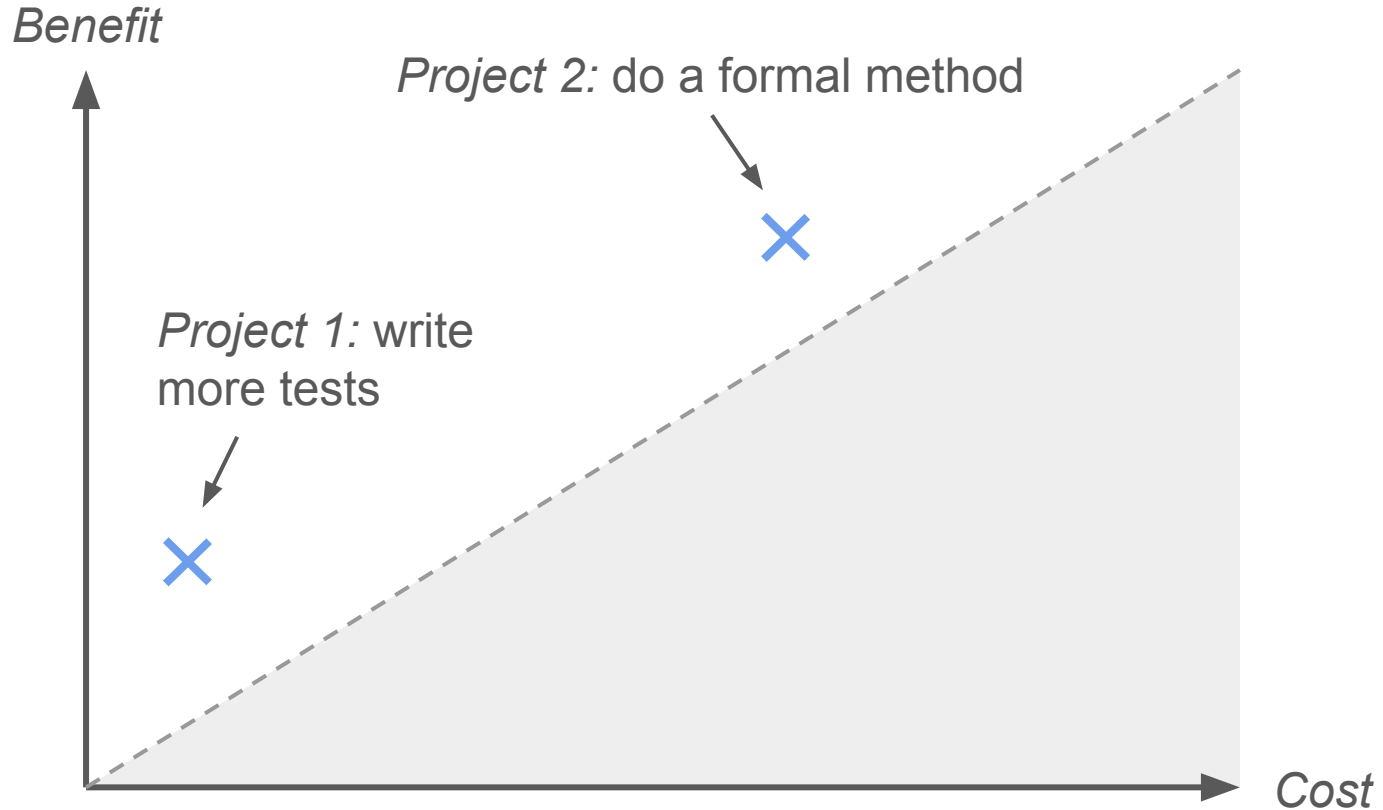
Many *many* systems
don't do these things

- Code review/Documentation
- Testing
- CI/CD
- Fuzzing/property based testing
- Modelling/Model Checking
- Symbolic Testing
- Program proof/Correct by Construction

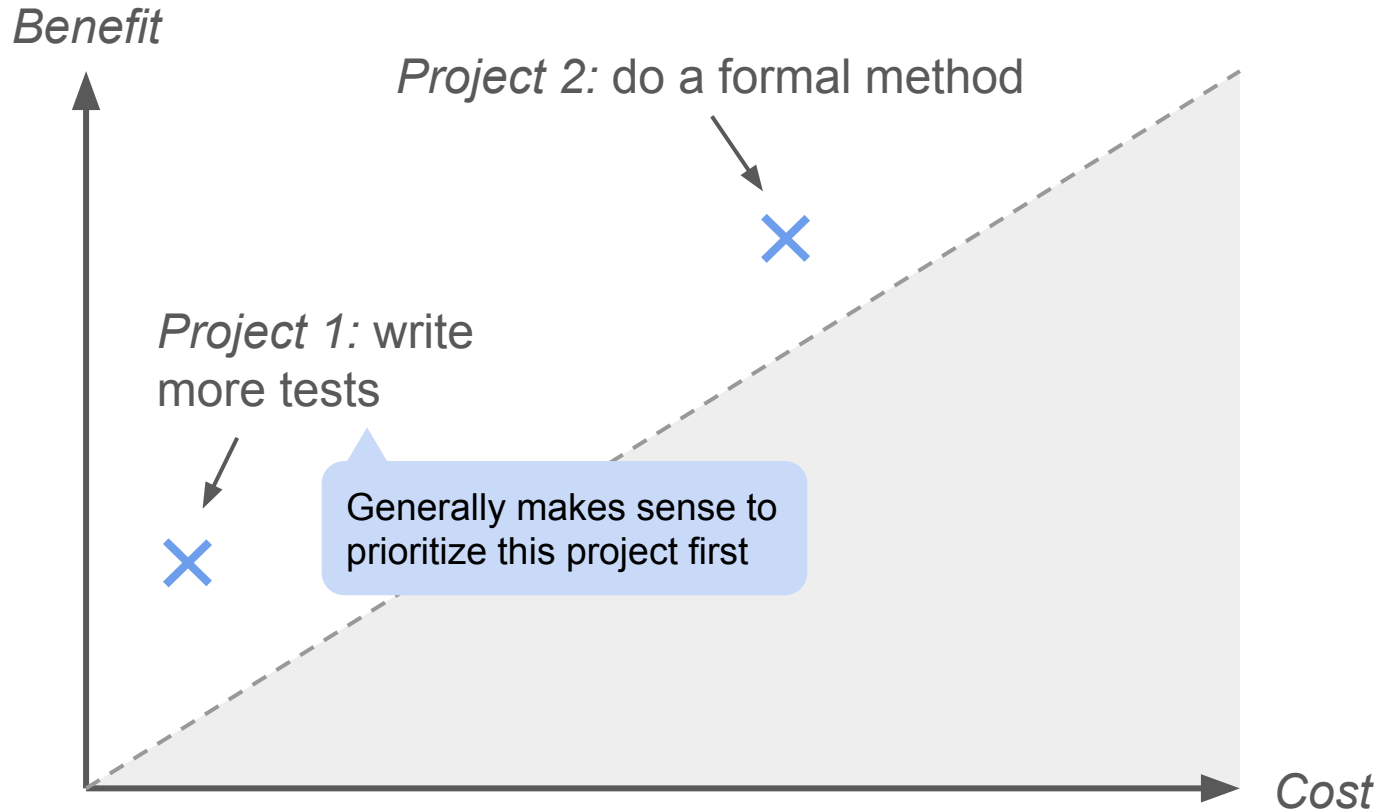


*Increasing effort,
Increasing confidence*

Cheap predictable > expensive risky projects



Cheap predictable > expensive risky projects



Strategy 1: “gold plating”

“Formal methods should be applied *after* conventional techniques”

- This approach makes sense
- I think it describes a lot of FM projects right now
- It really limits the number of projects that pencil out

Strategy 2: YOLO

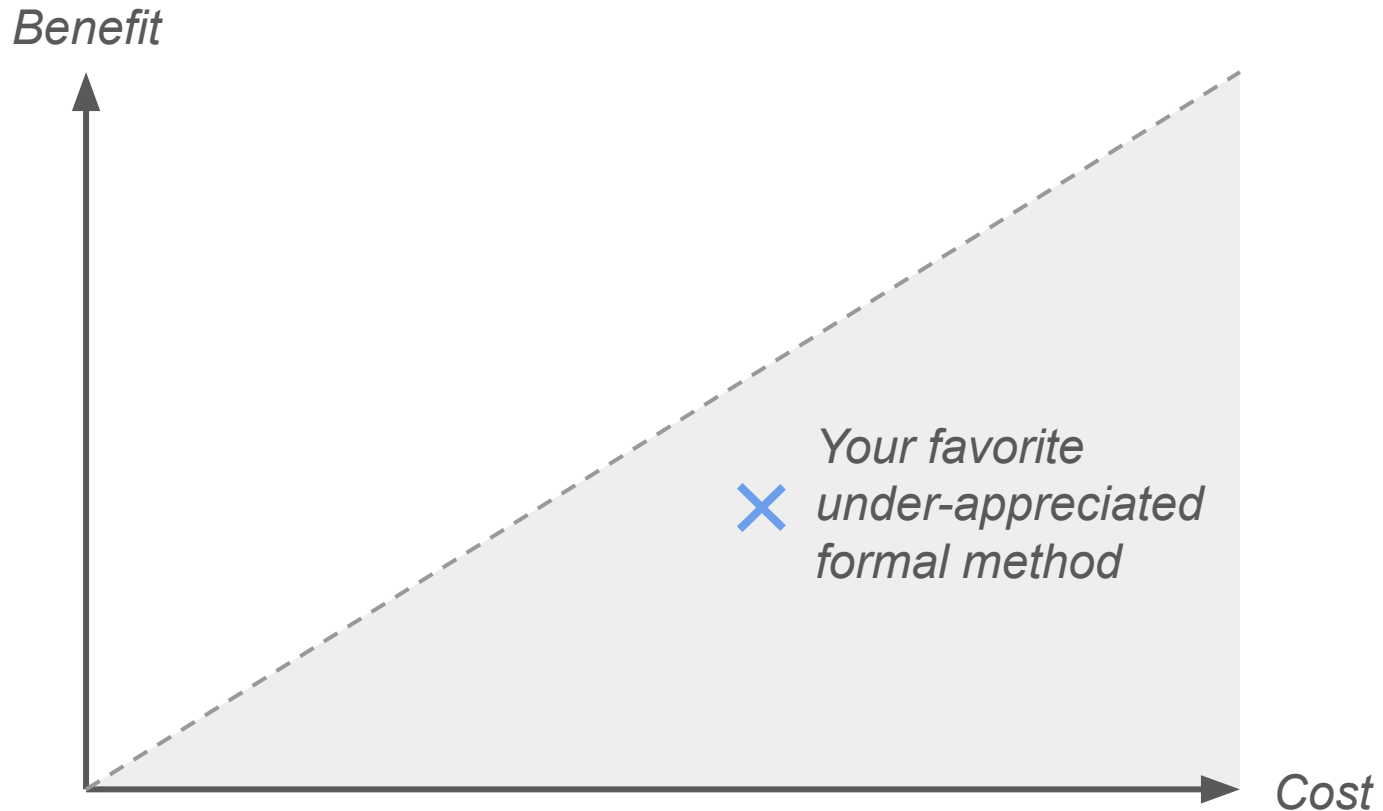
“Formal methods *can replace* conventional techniques”

- This sounds great!
- Really mostly not true right now
- Some domains where it's been successful (eg. Tiros / Zelkova @ AWS)

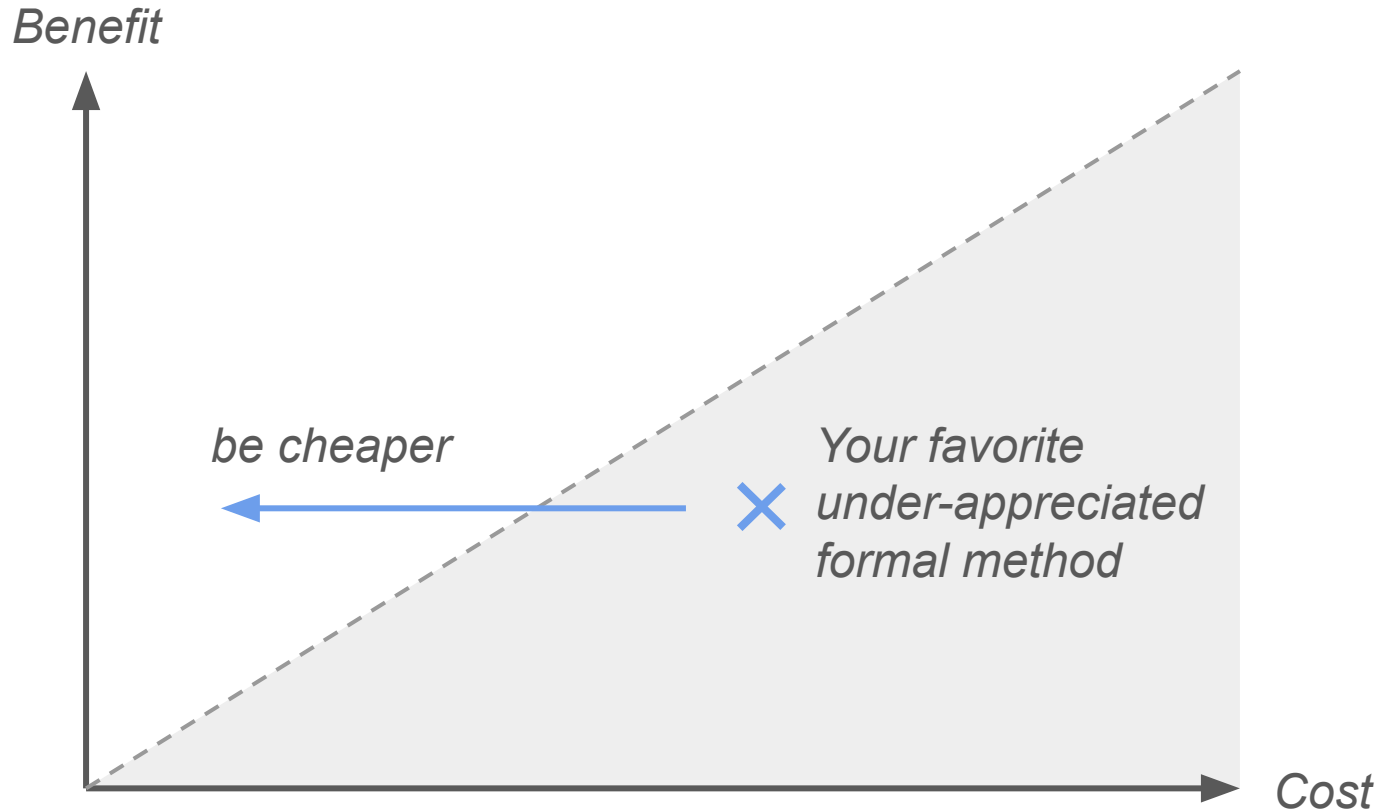
Hot takes:

- Project have to deliver value early
- Correctness often doesn't matter
- Specifications don't exist
- It's hard to define & explain success
- Do cheap things first

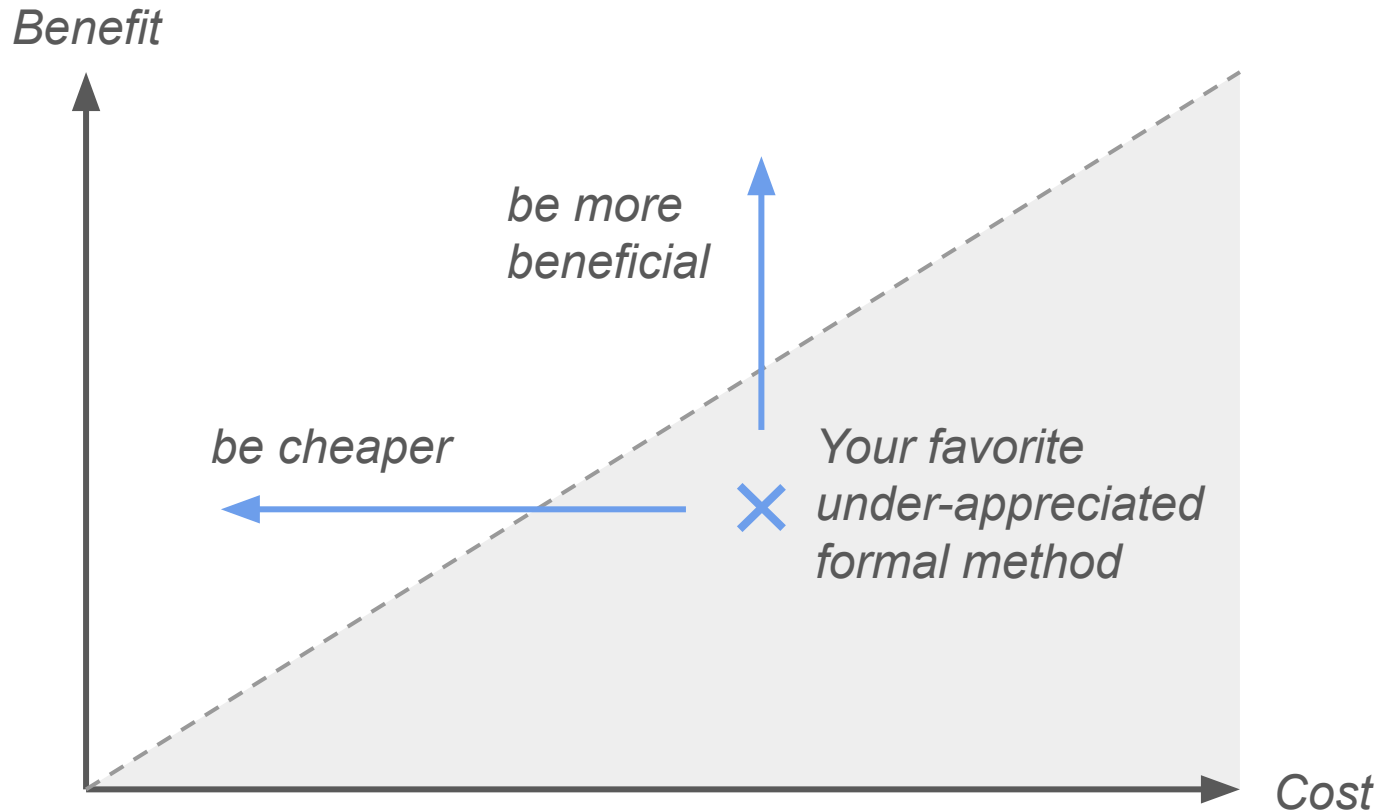
Theme: costs vs. benefits



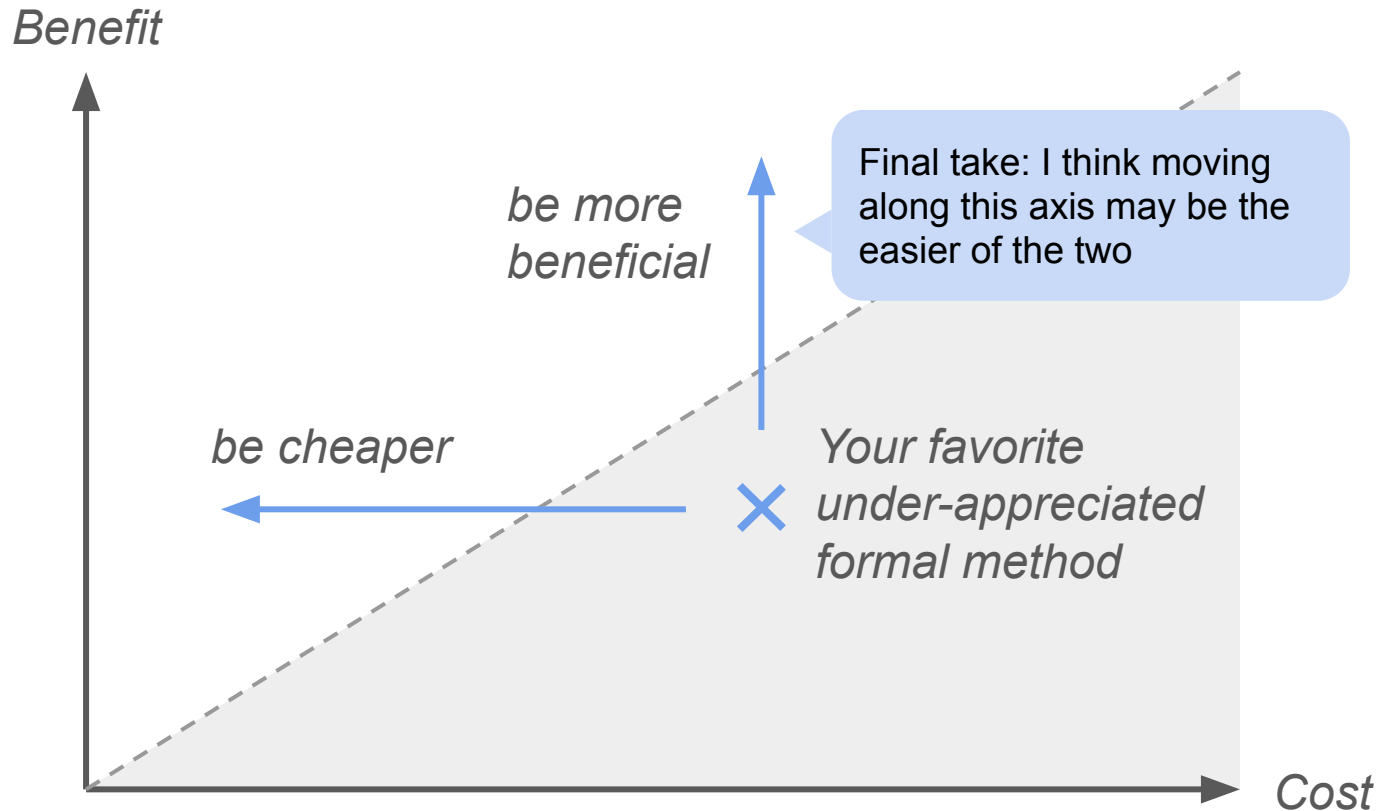
Theme: costs vs. benefits



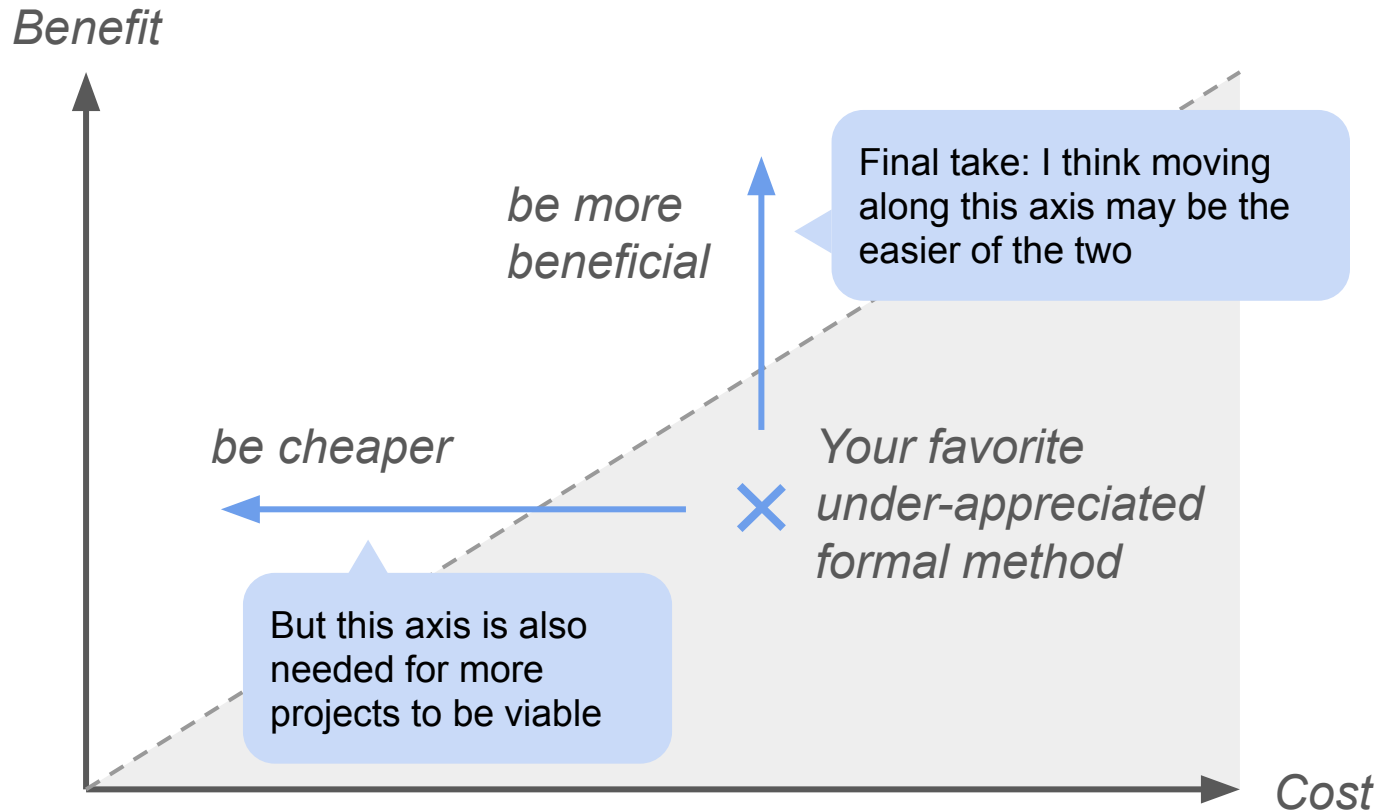
Theme: costs vs. benefits



Theme: costs vs. benefits



Theme: costs vs. benefits



Thanks!

Mike Dodds

miked@galois.com

| galois |