Microservices

scenarios of the near and far future

Saverio Giallorenzo

Howdy

Saverio

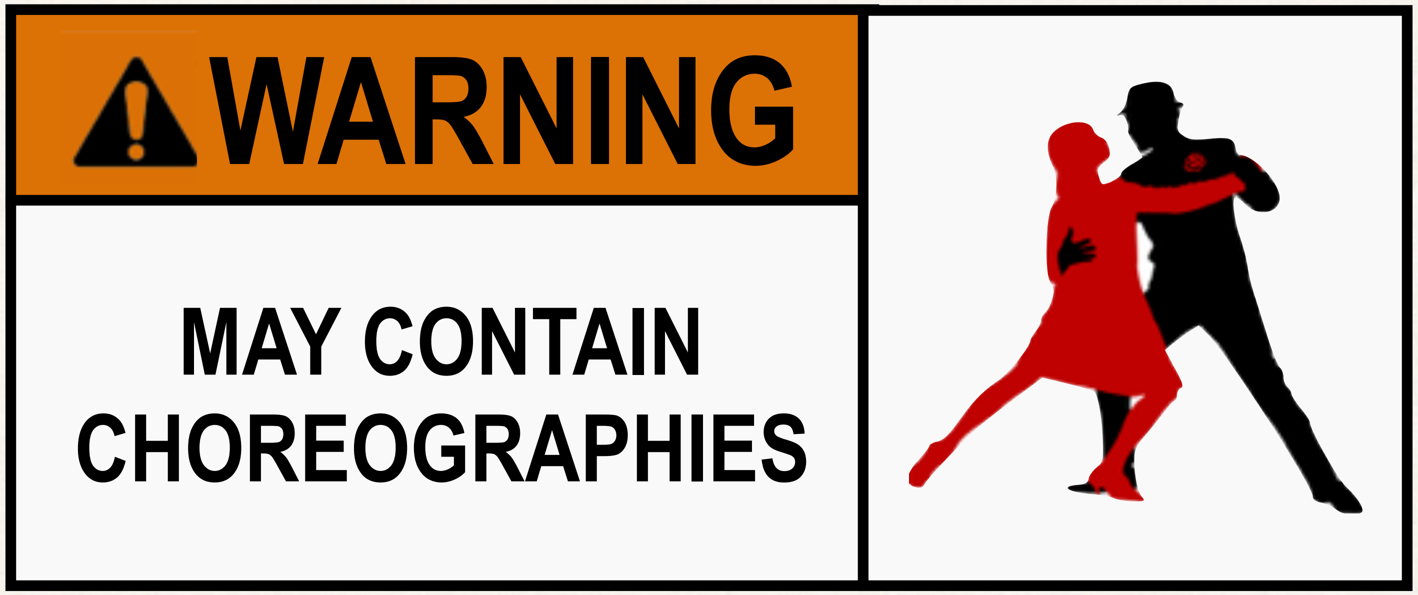
Post-doc at the **Department of Computer Science and Engineering** of **University of Bologna**.

Research topics:

- Concurrent and distributed programming;
- Choreographies, Session
- Types and Process Algebras.
- Microservices;
- Jolie;

Microservices

scenarios of near and far future

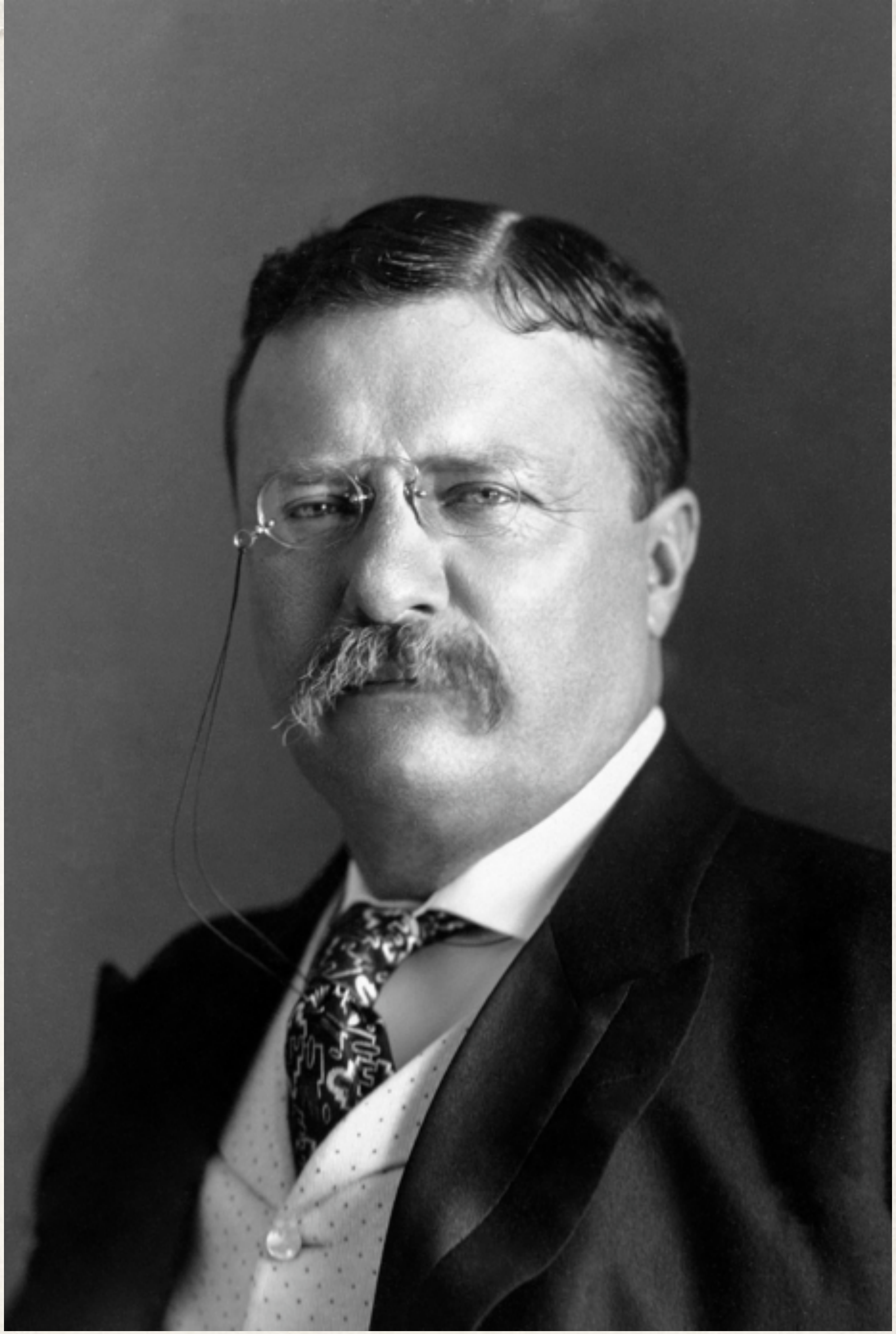


Saverio Giallorenzo

Today's Limits

There is no effort without error and shortcoming.

"Citizenship in a Republic", Theodore Roosevelt, 1910

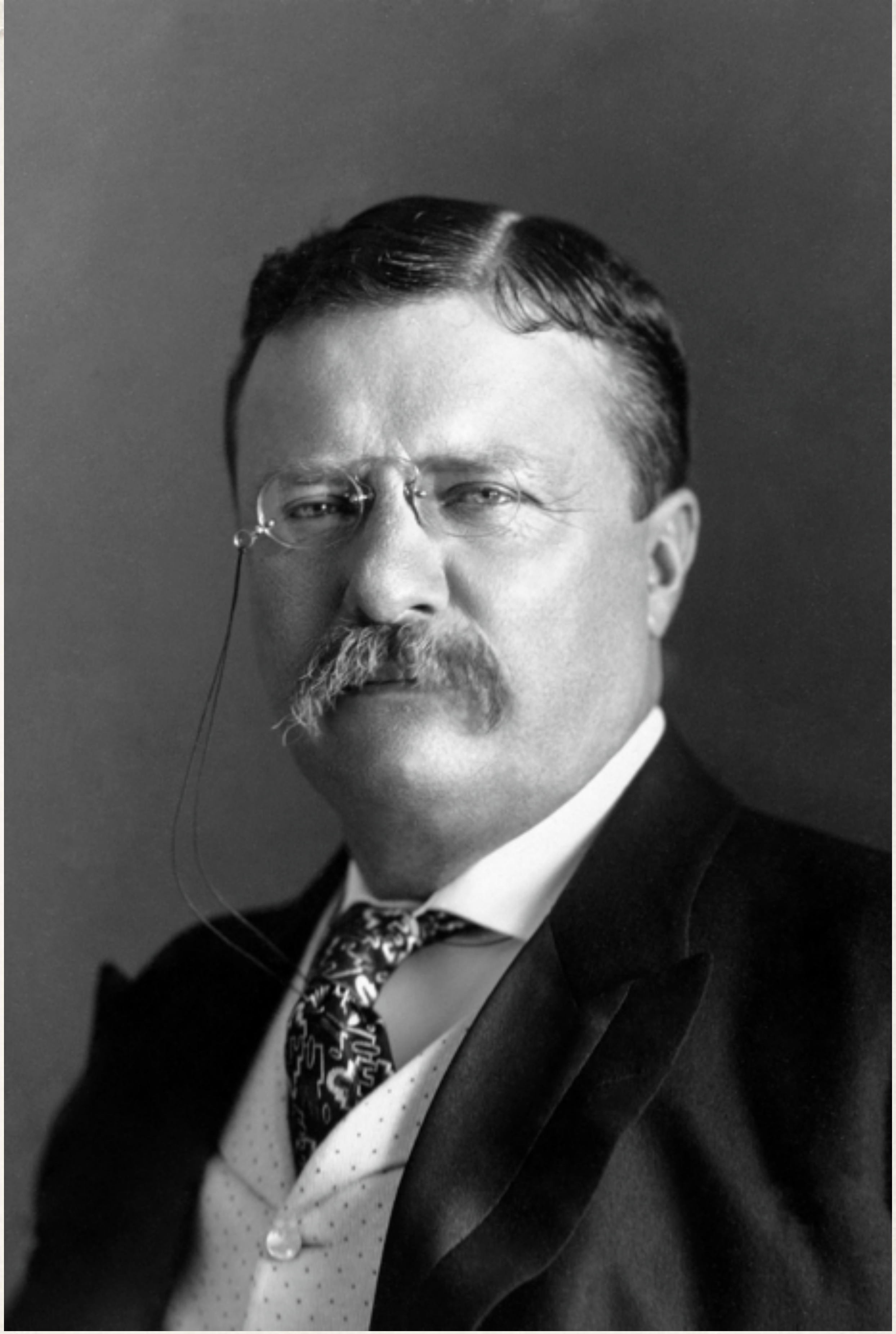


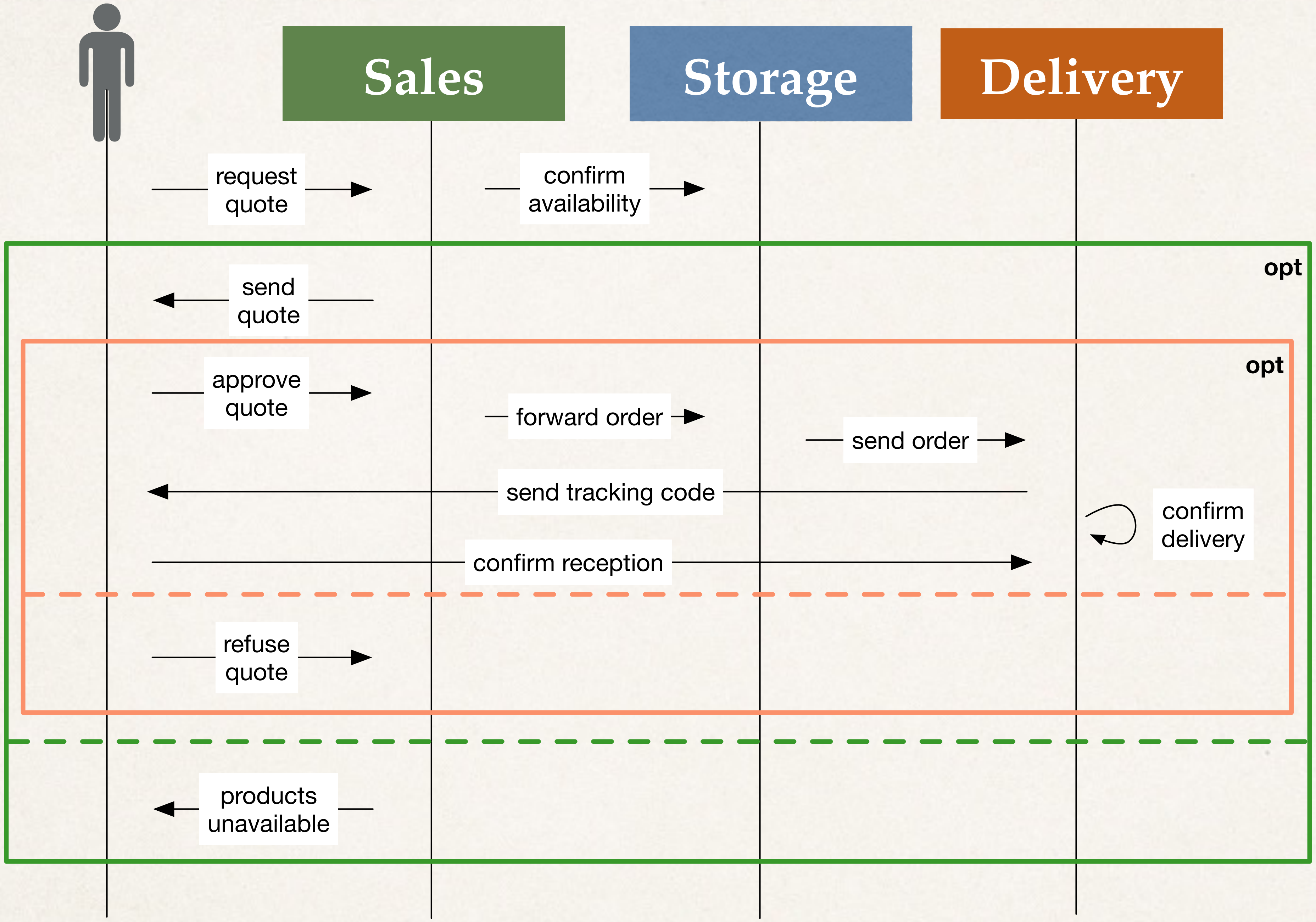
Today's Limits

innovation

There is no ~~effort~~ without error and shortcoming.

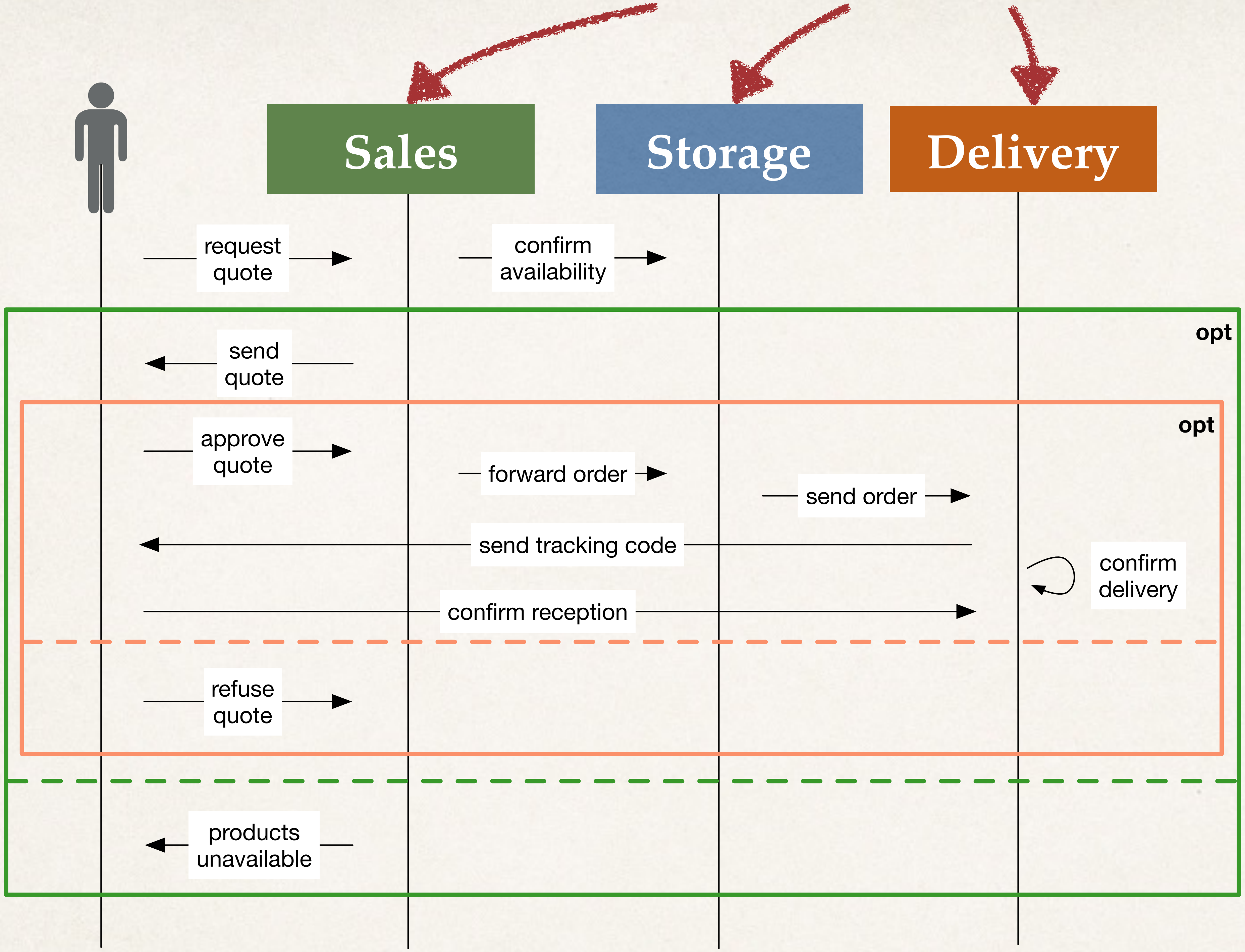
"Citizenship in a Republic", Theodore Roosevelt, 1910





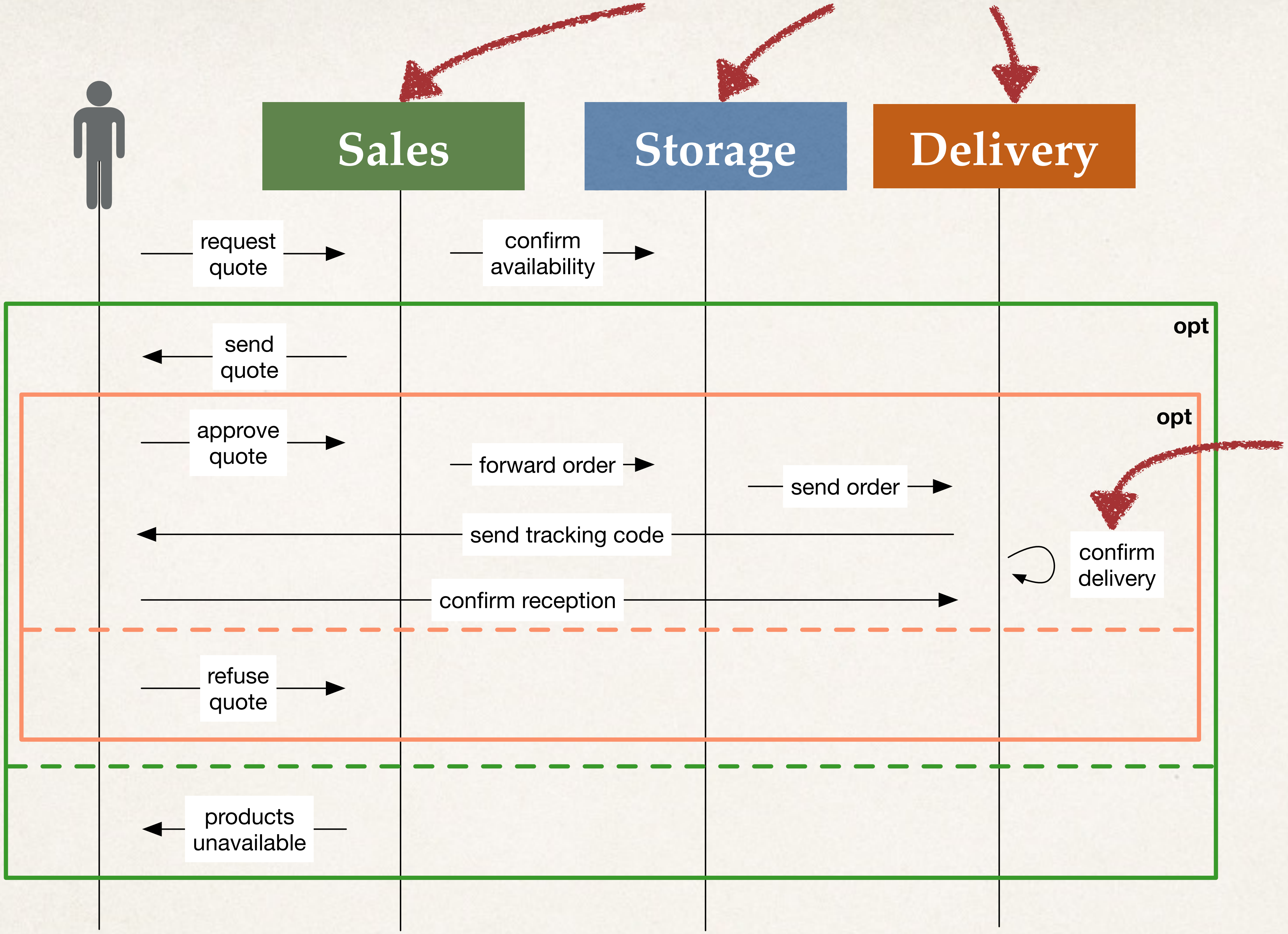
Distributed Programming

What's here inside (e.g., error tracing)?



Distributed Programming

What's here inside (e.g., error tracing)?

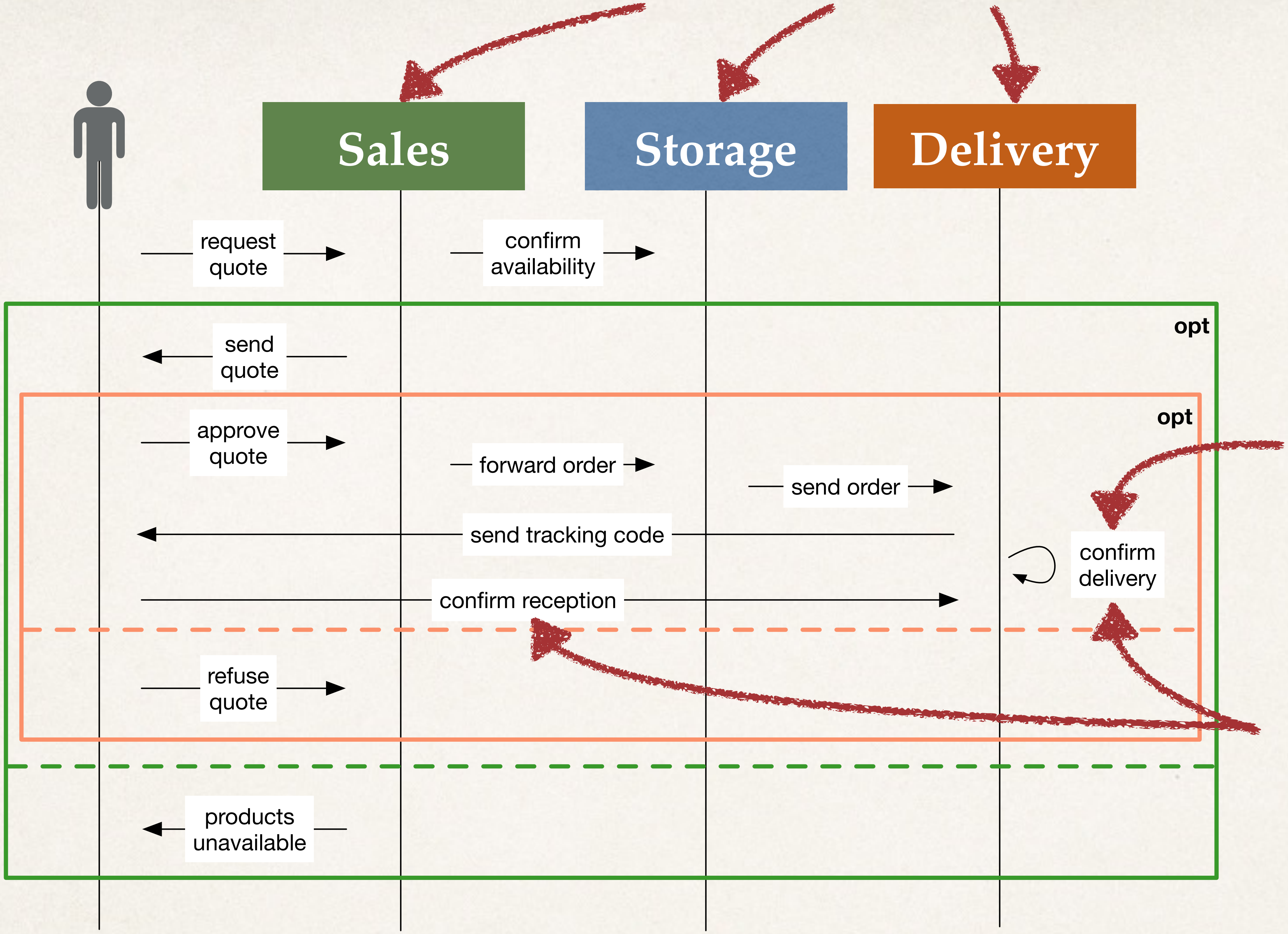


Internal functionality? Does the deliverer provide it? Docs/APIs?

Distributed Programming

Distributed Programming

What's here inside (e.g., error tracing)?



Sales

Storage

Delivery

request quote

confirm availability

opt

send quote

opt

approve quote

forward order

send order

send tracking code

confirm delivery

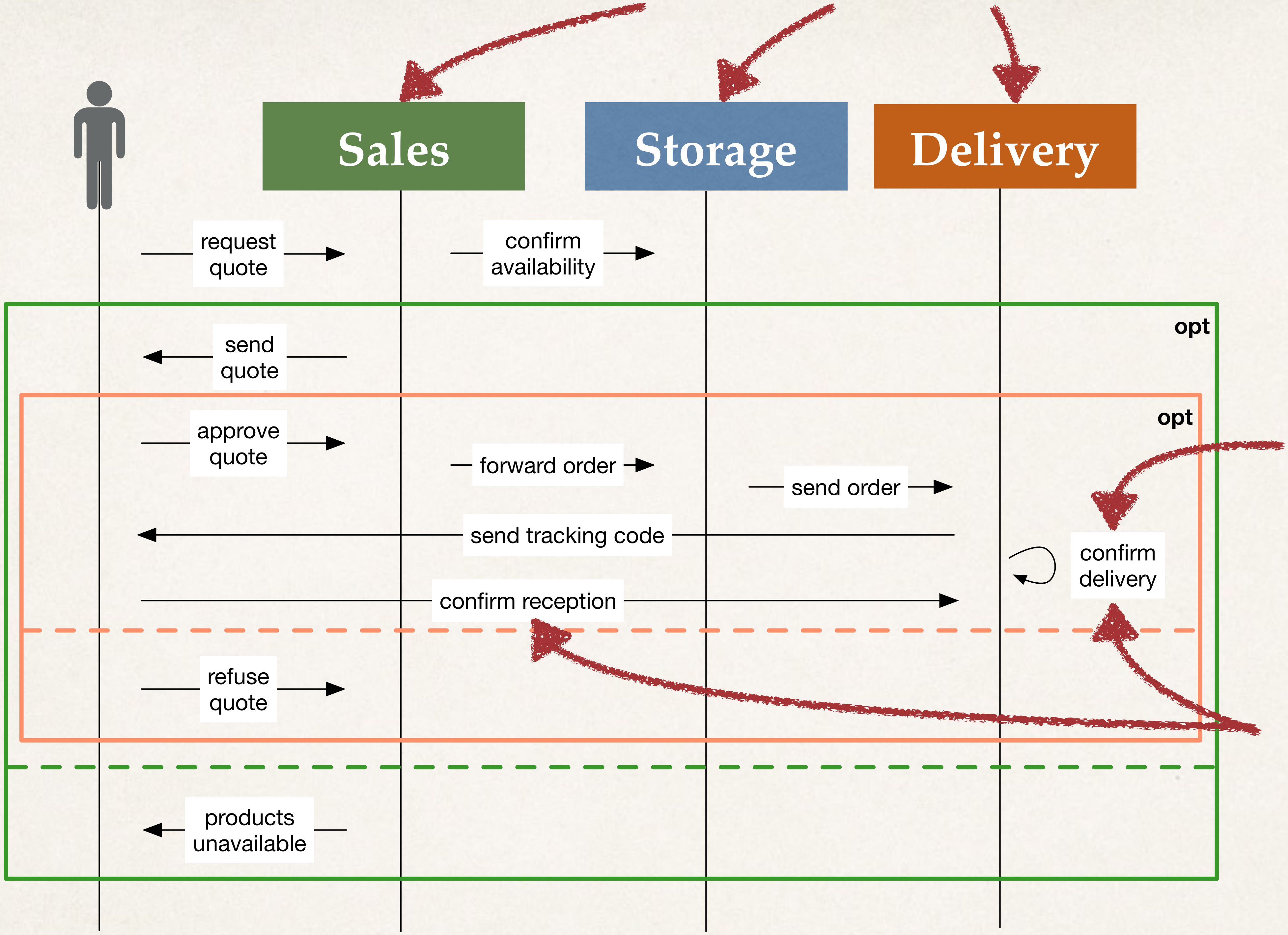
confirm reception

refuse quote

products unavailable

Internal functionality? Does the deliverer provide it? Docs / APIs?

What's here inside (e.g., error tracing)?



Internal functionality? Does the deliverer provide it? Docs/APIs?

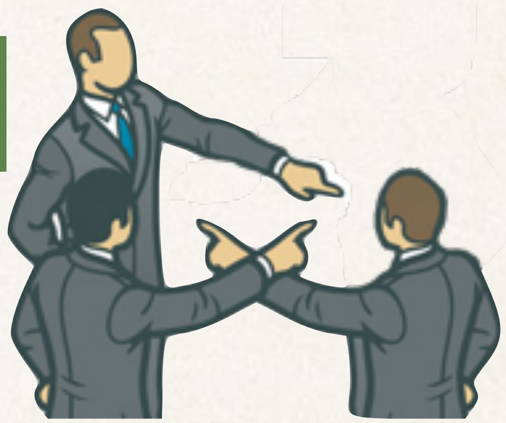
Sequential (in which order) or in parallel?

Distributed Programming

Direction

Sales

Delivery



Storage

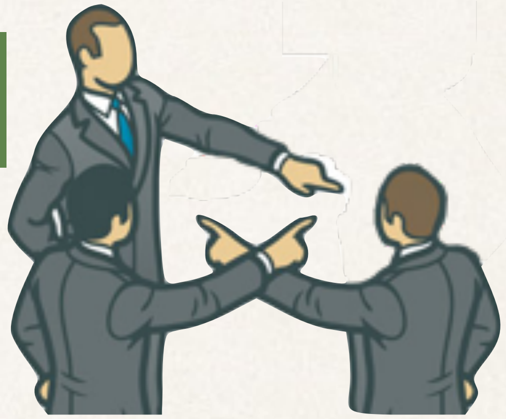
“Not my problem”

Direction

Big Picture

Sales

Delivery



Storage

“Not my problem”

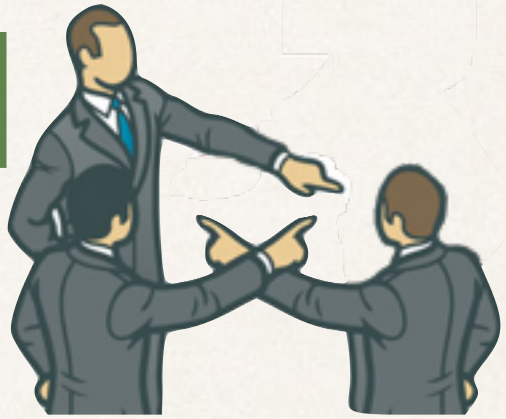
Direction

Big Picture

Gulf of execution

Sales

Delivery



Storage

“Not my problem”

Direction

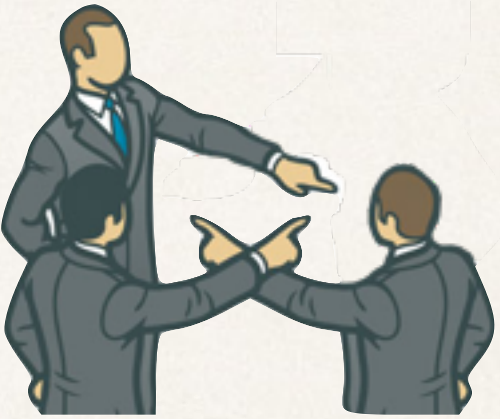
Big Picture

Gulf of execution

Micro-management

Sales

Delivery



Storage

“Not my problem”

Direction

Big Picture

Gulf of execution

Micro-management

Coordination?

Accountability?

Sales

Delivery

Storage

“Not my problem”

Direction

Big Picture

Gulf of execution

Gulf of Evaluation

Micro-management

Coordination?

Accountability?

Sales

Delivery

Storage

“Not my problem”

Direction

Big Picture

Gulf of execution

Gulf of Evaluation

Micro-management

Coordination?

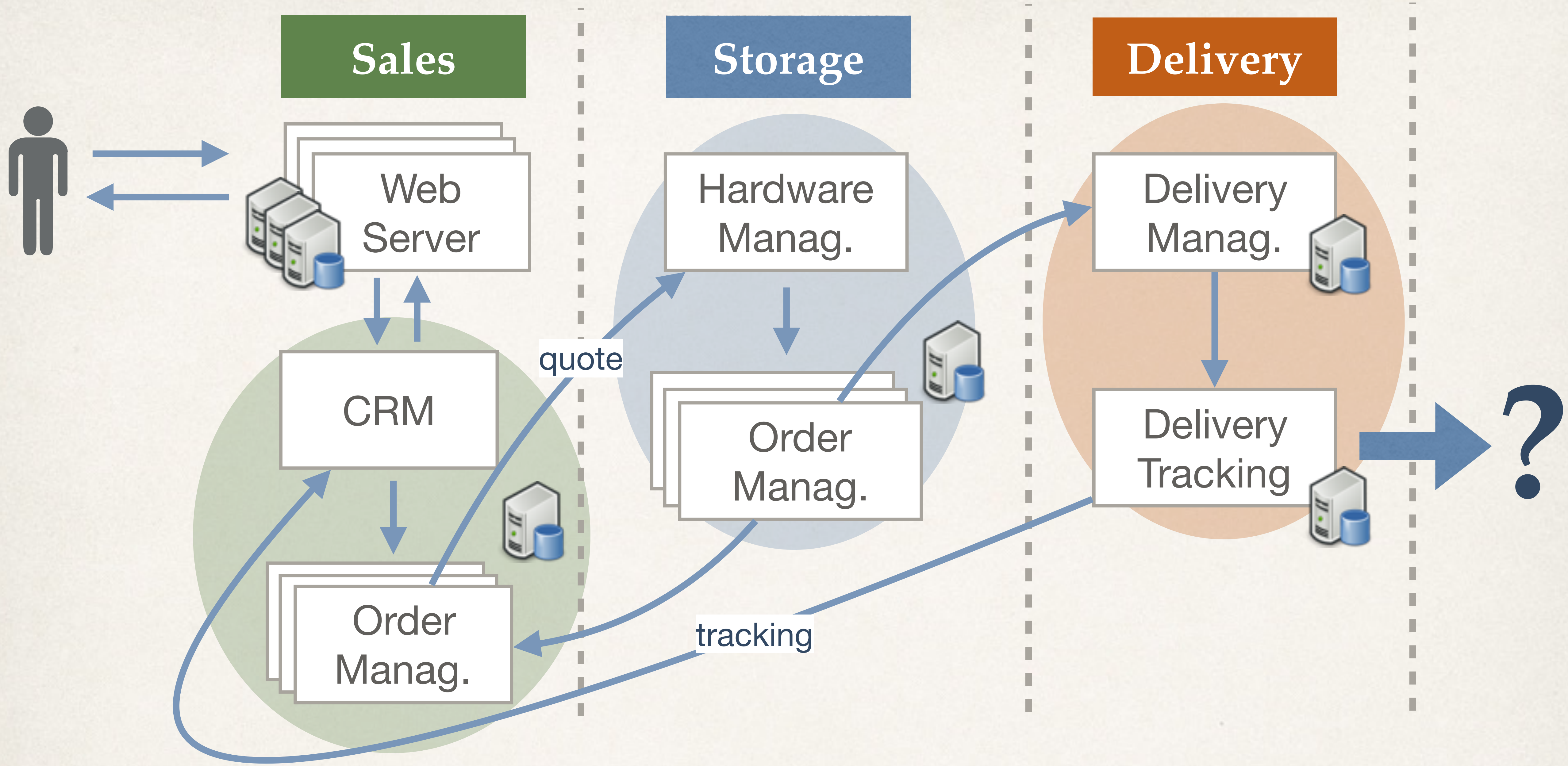
Accountability?

Sales

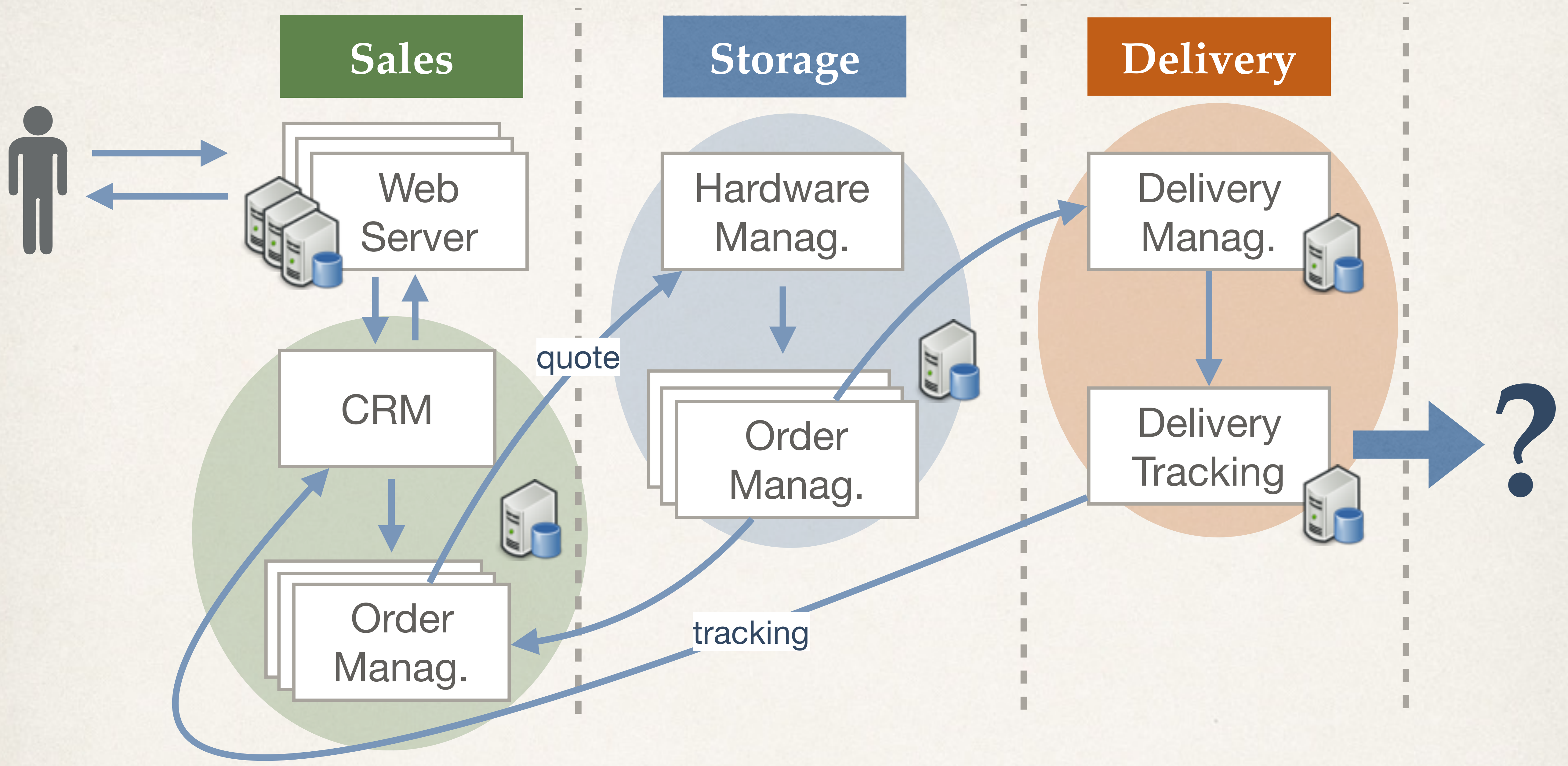
Delivery

Storage

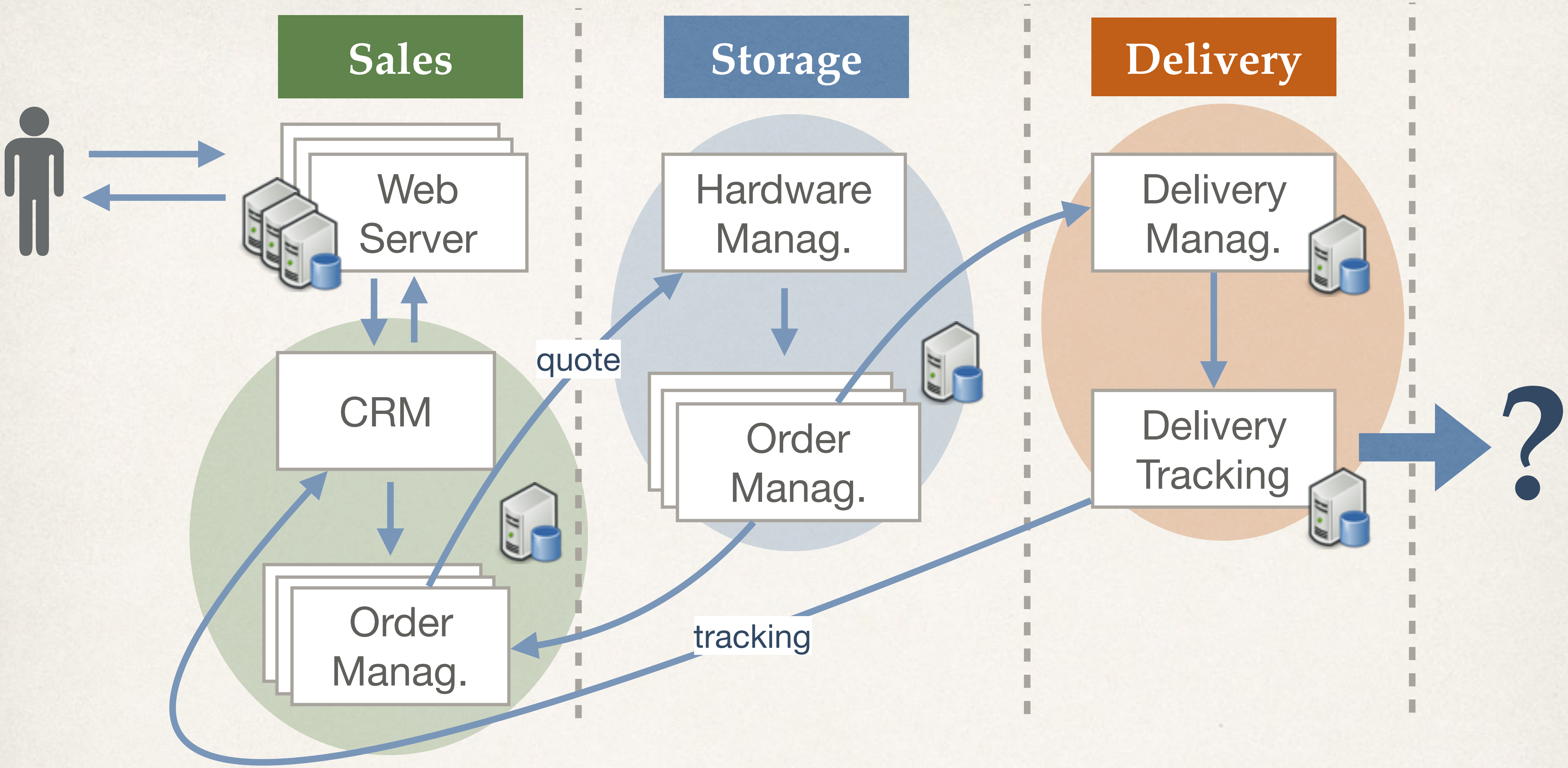
“Not my problem”



Scalable Architectures

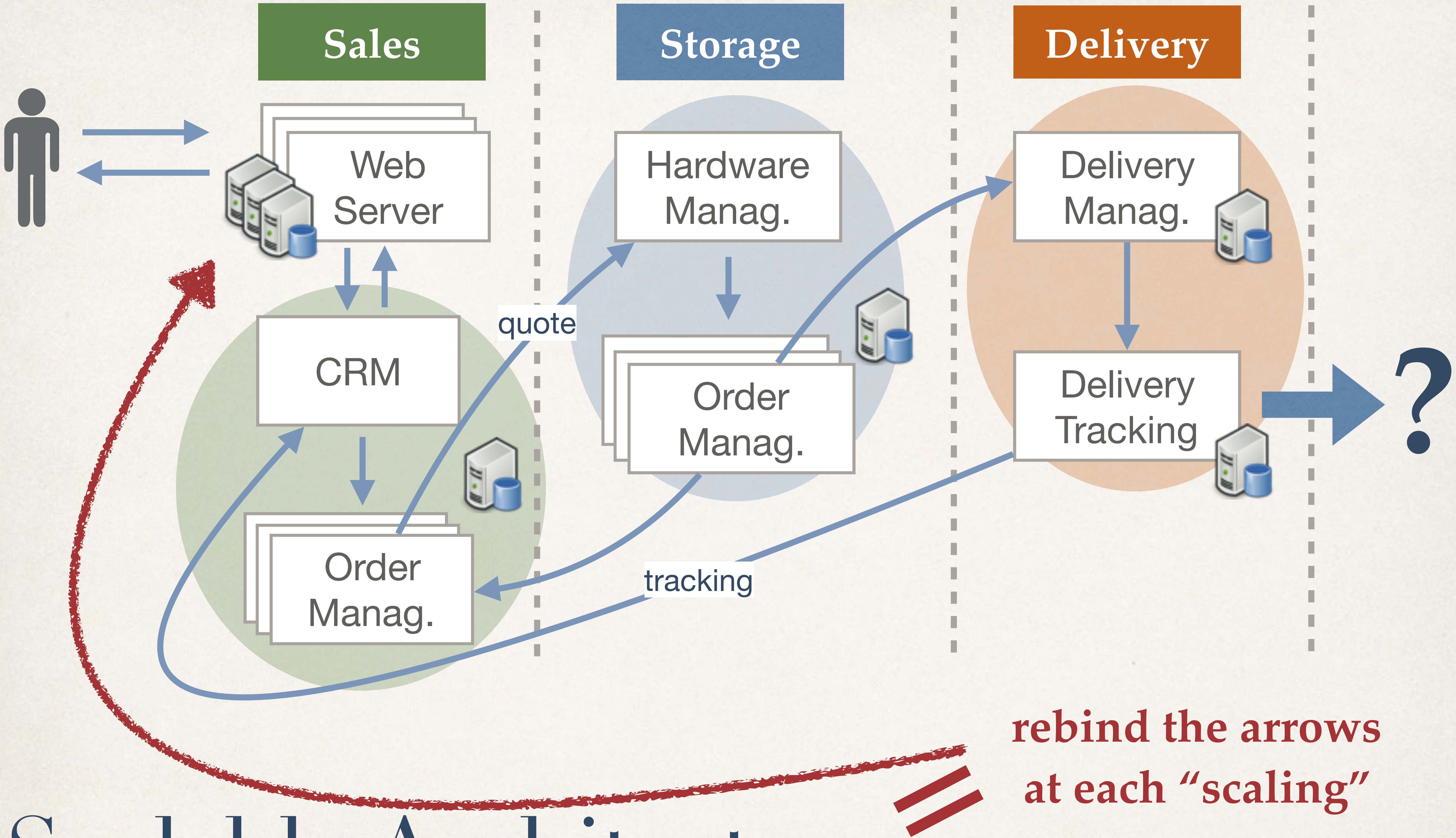


Scalable Architectures



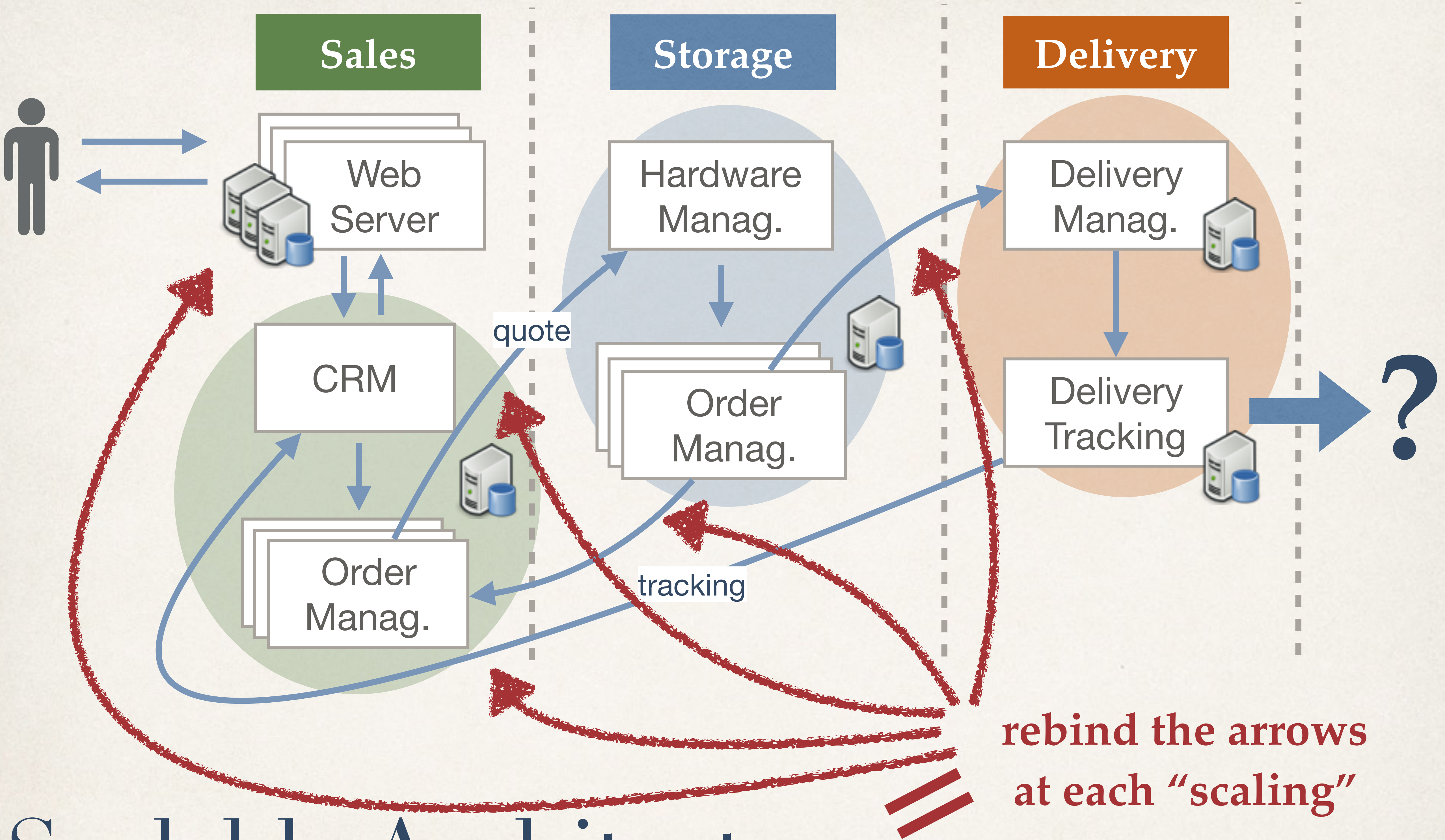
rebind the arrows at each "scaling"

Scalable Architectures



rebind the arrows at each "scaling"

Scalable Architectures



Sales

Storage

Delivery

Web Server

CRM

Order Manag.

quote

Hardware Manag.

Order Manag.

tracking

Delivery Manag.

Delivery Tracking

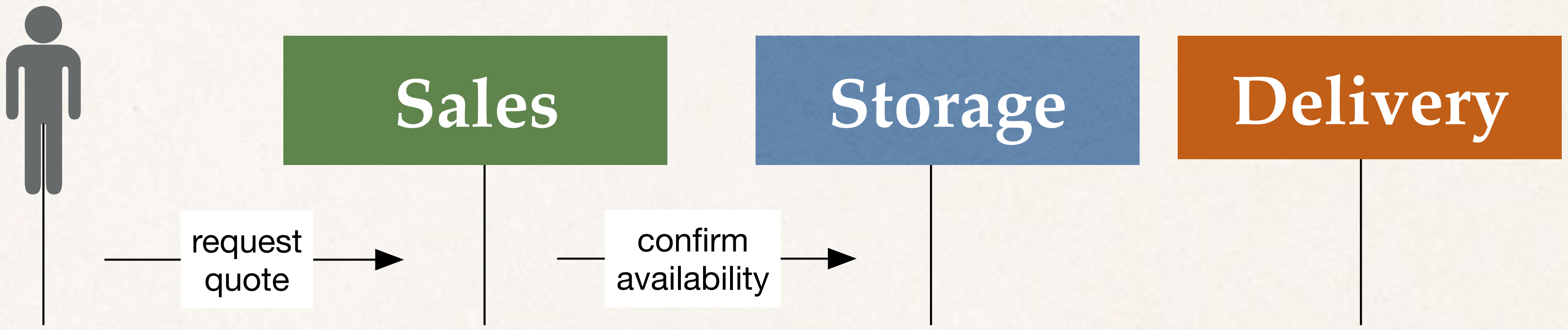
?

rebind the arrows at each “scaling”

Scalable Architectures

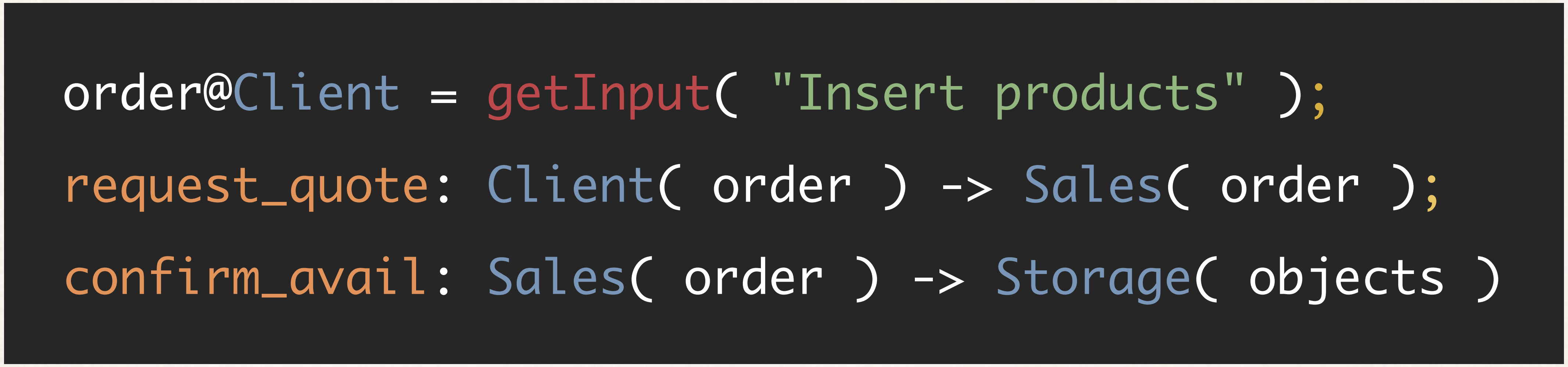
A look at the future
Choreographic Programming

Enter AIOCJ



```
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
```

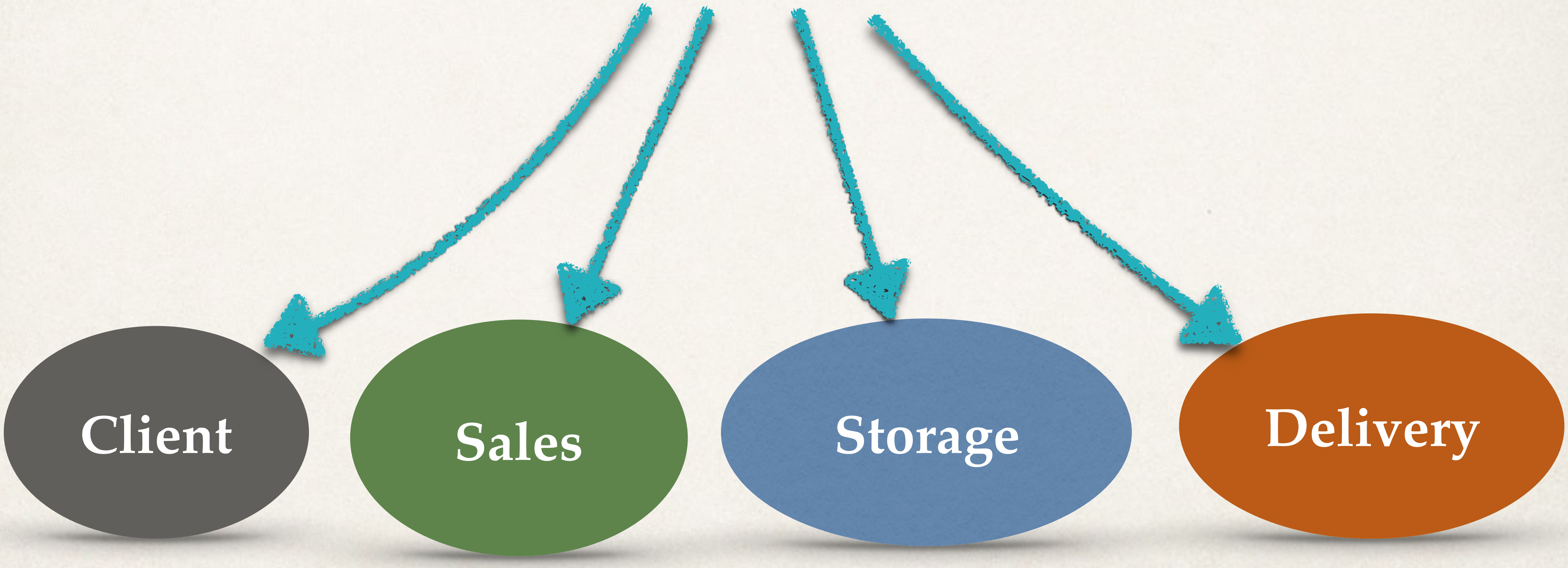
Architectural Vision



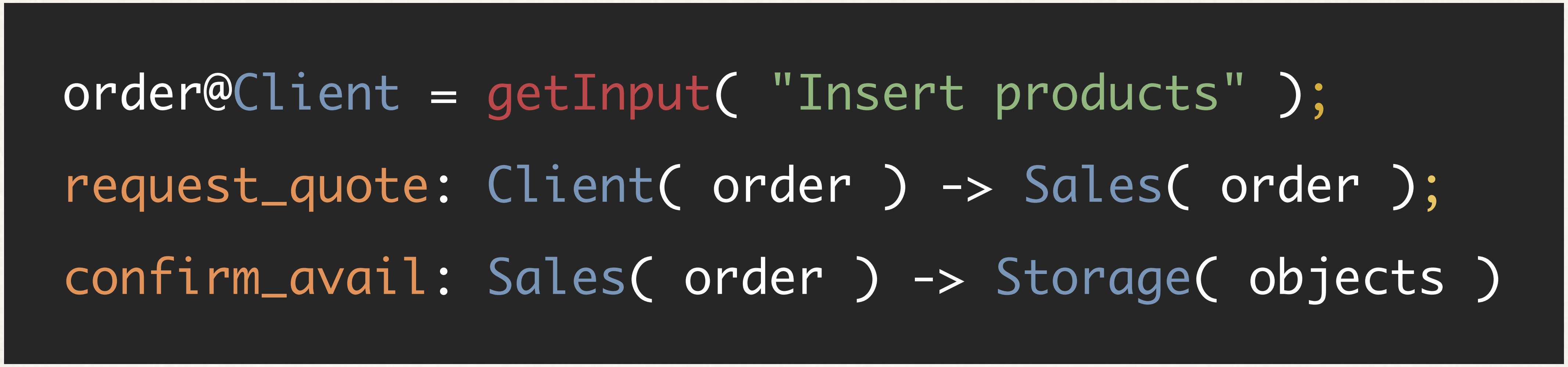
```
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
```

Architectural Vision

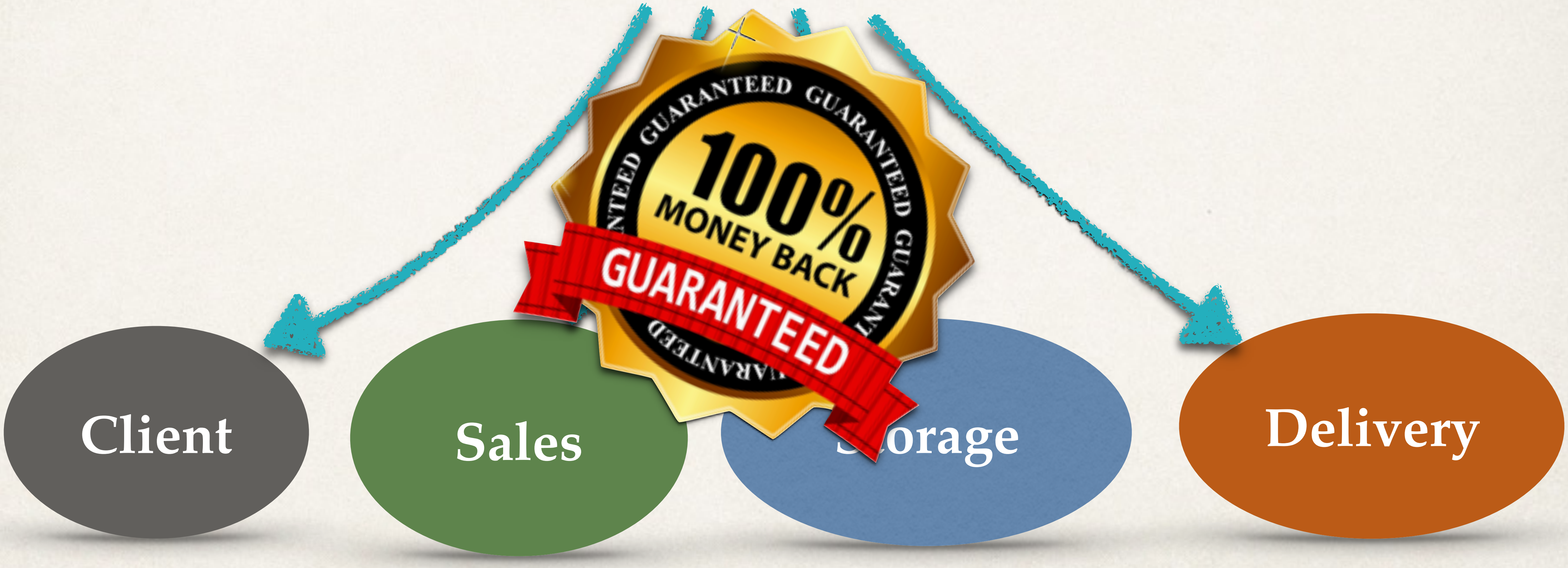
```
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
```

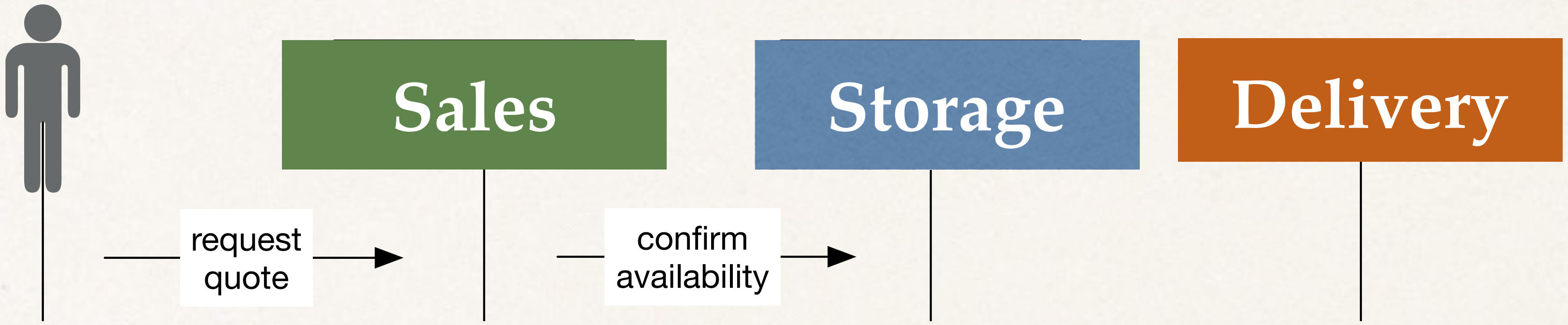


Architectural Vision

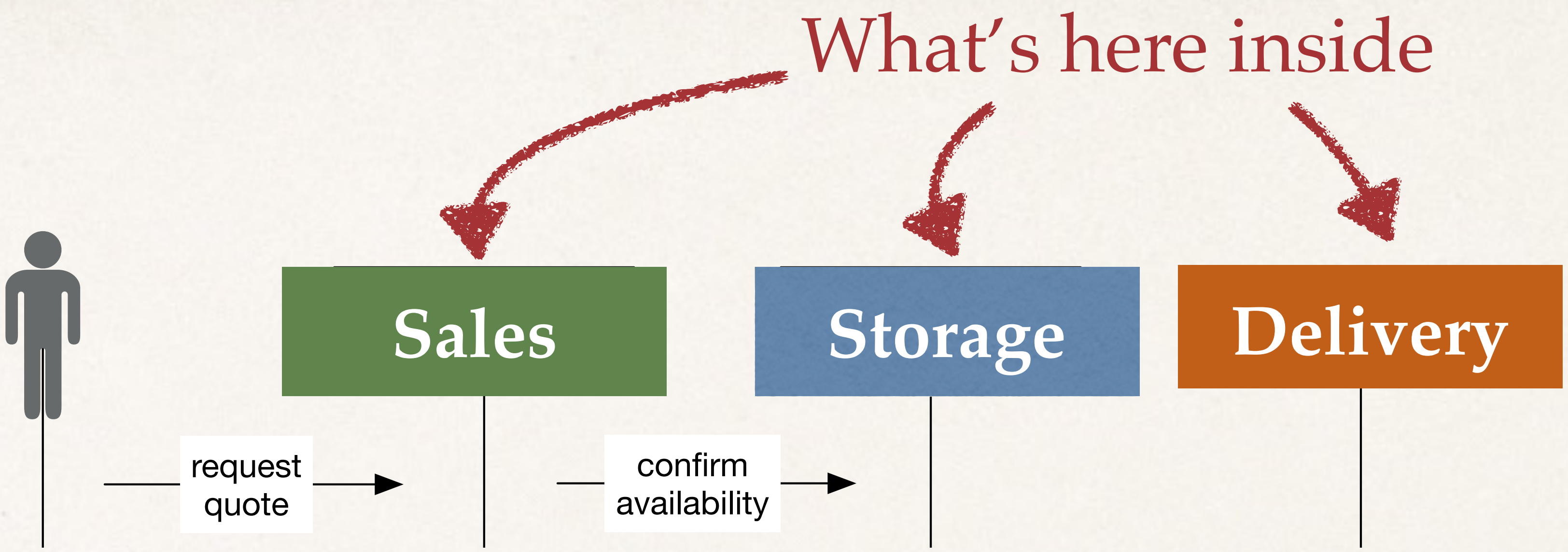


```
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
```

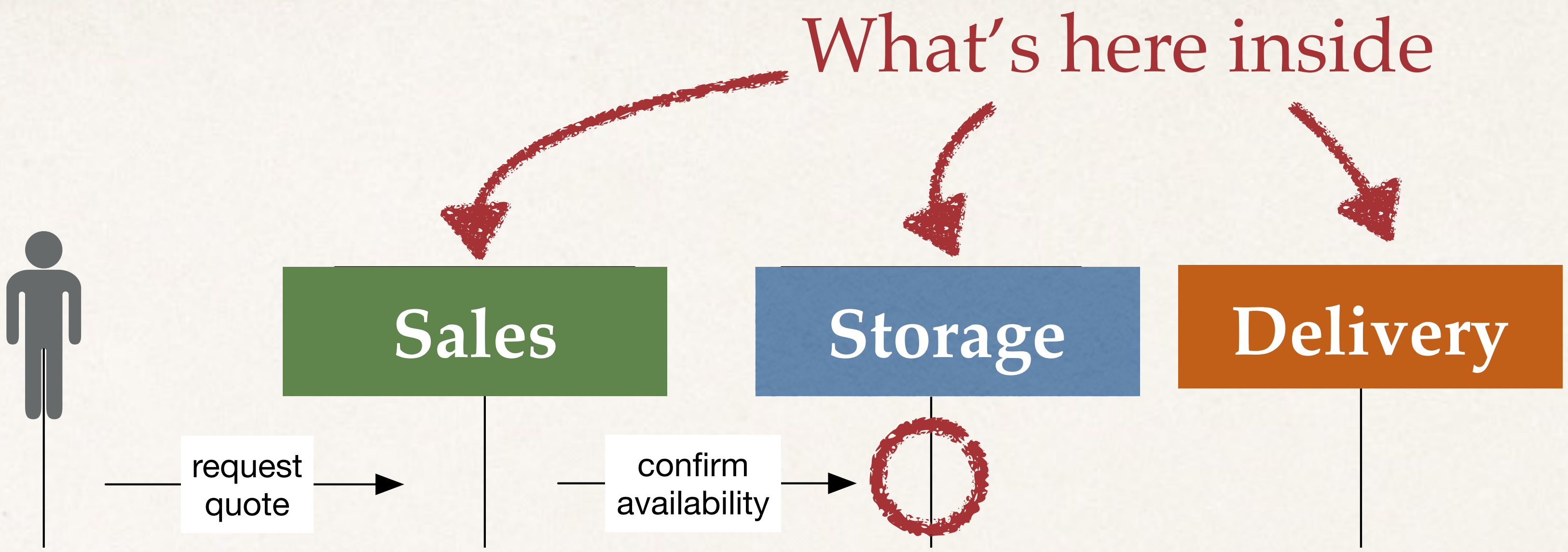




```
include checkAvail from “socket://storage:8000"


order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
```

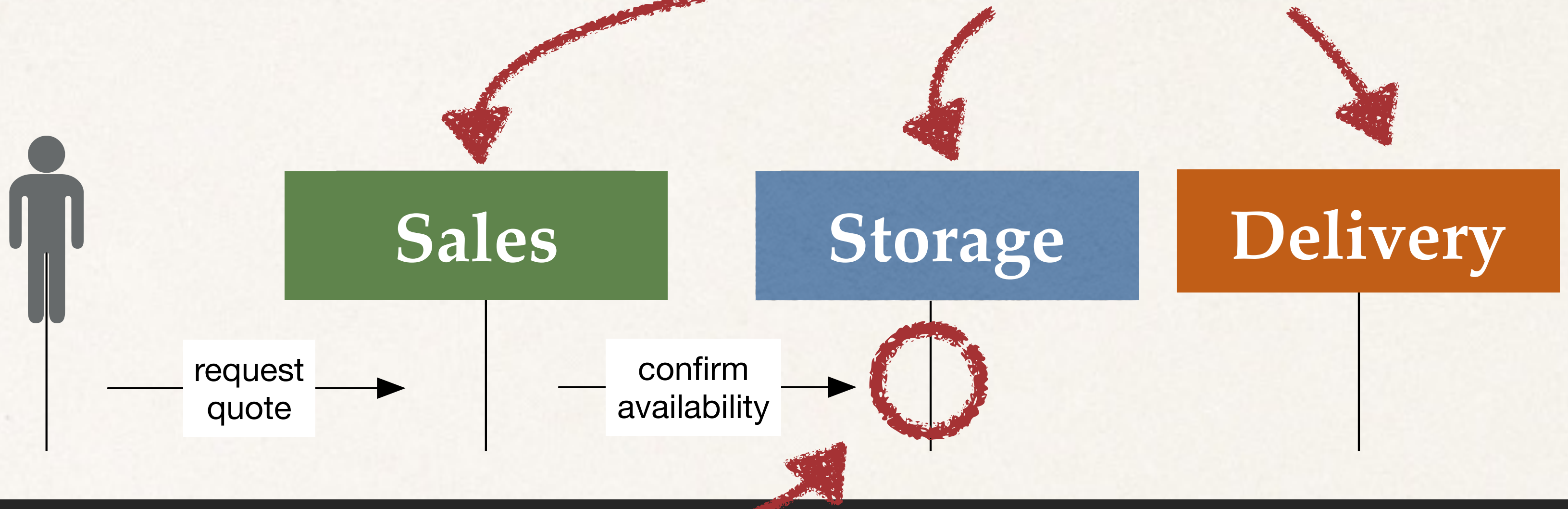


```
include checkAvail from “socket://storage:8000"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
```



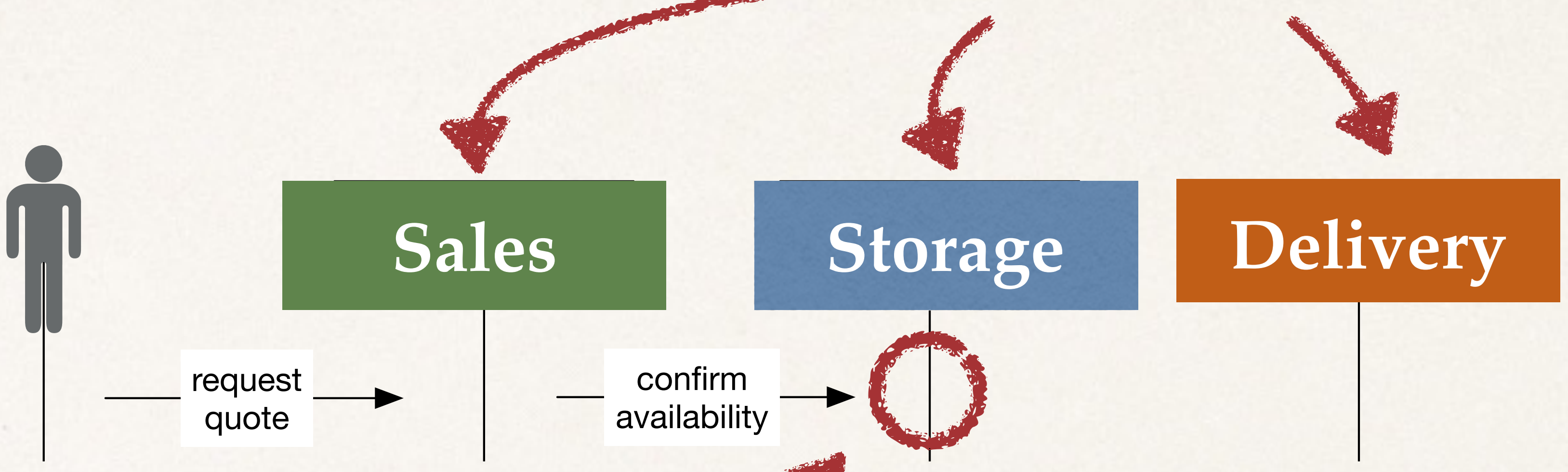
```
include checkAvail from “socket://storage:8000"


order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
```

What's here inside



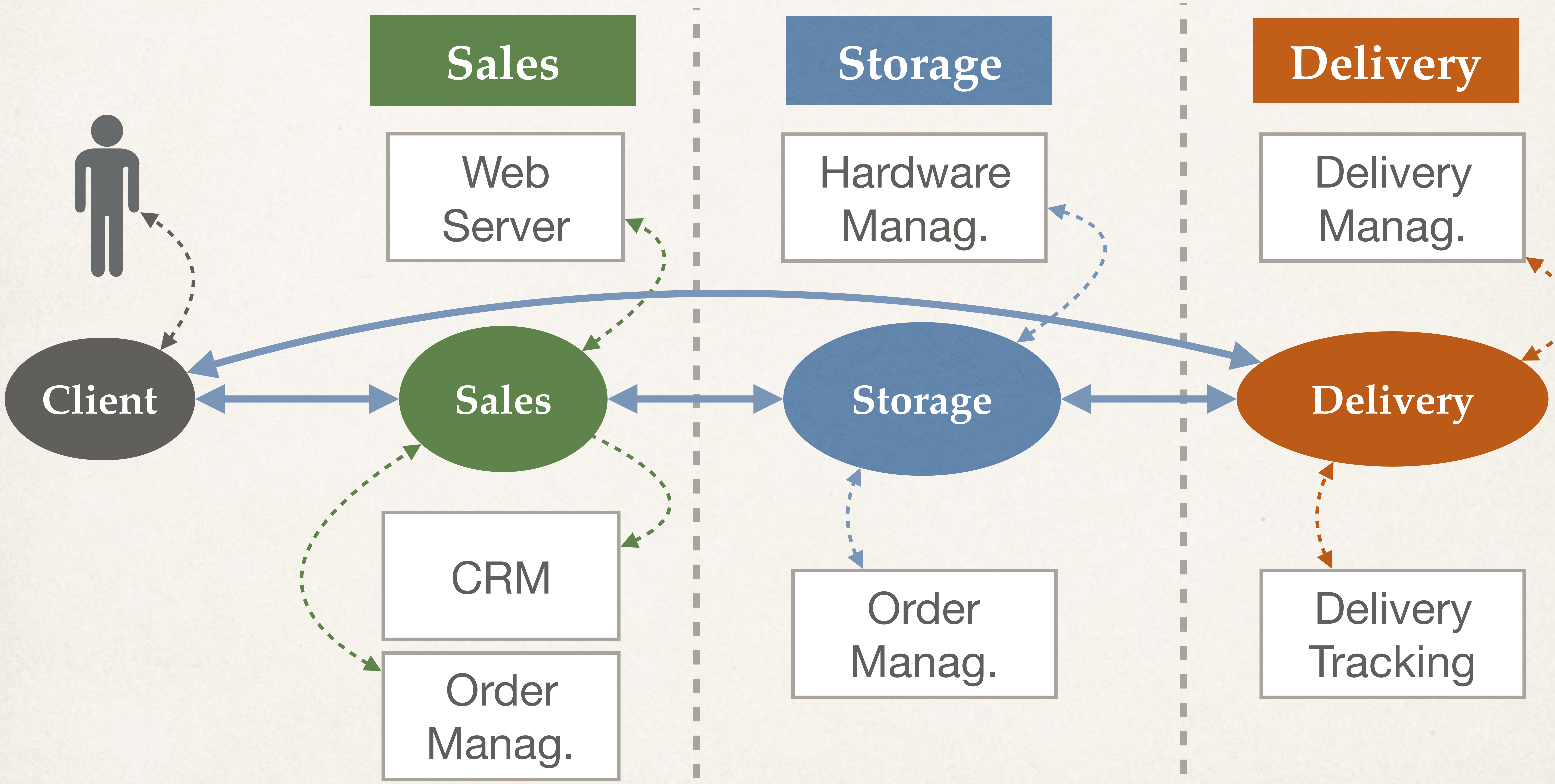
```
include checkAvail from “socket://storage:8000"


order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
```

What’s here inside

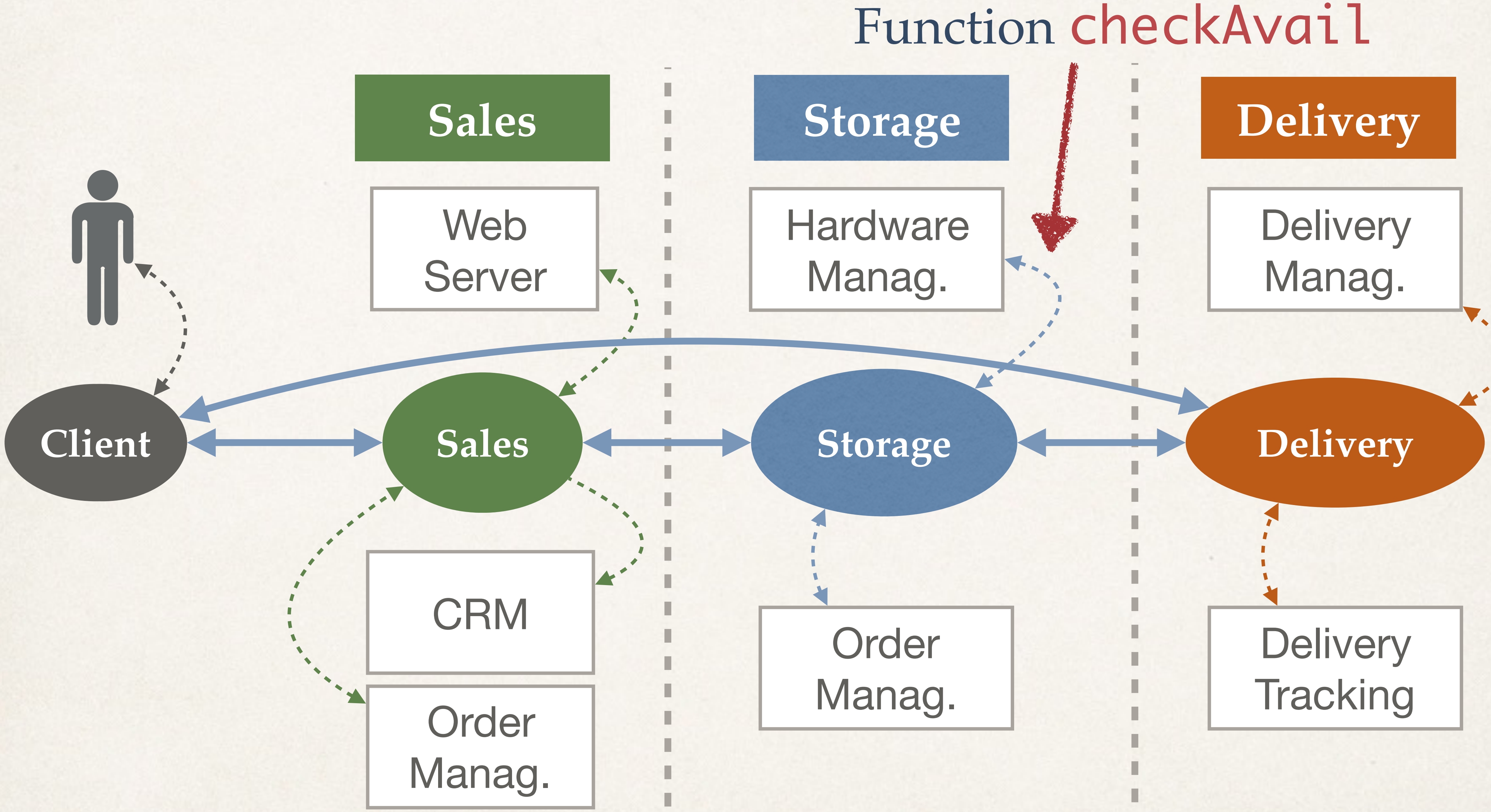


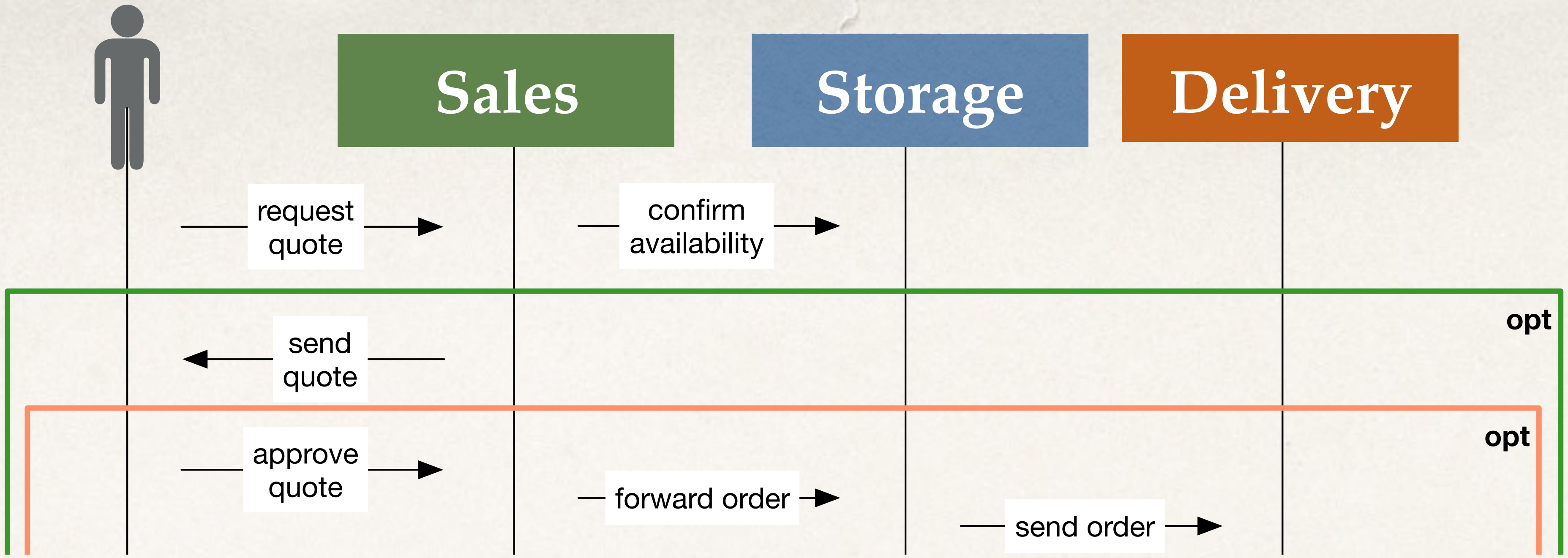
```
include checkAvail from “socket://storage:8000"


order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
```

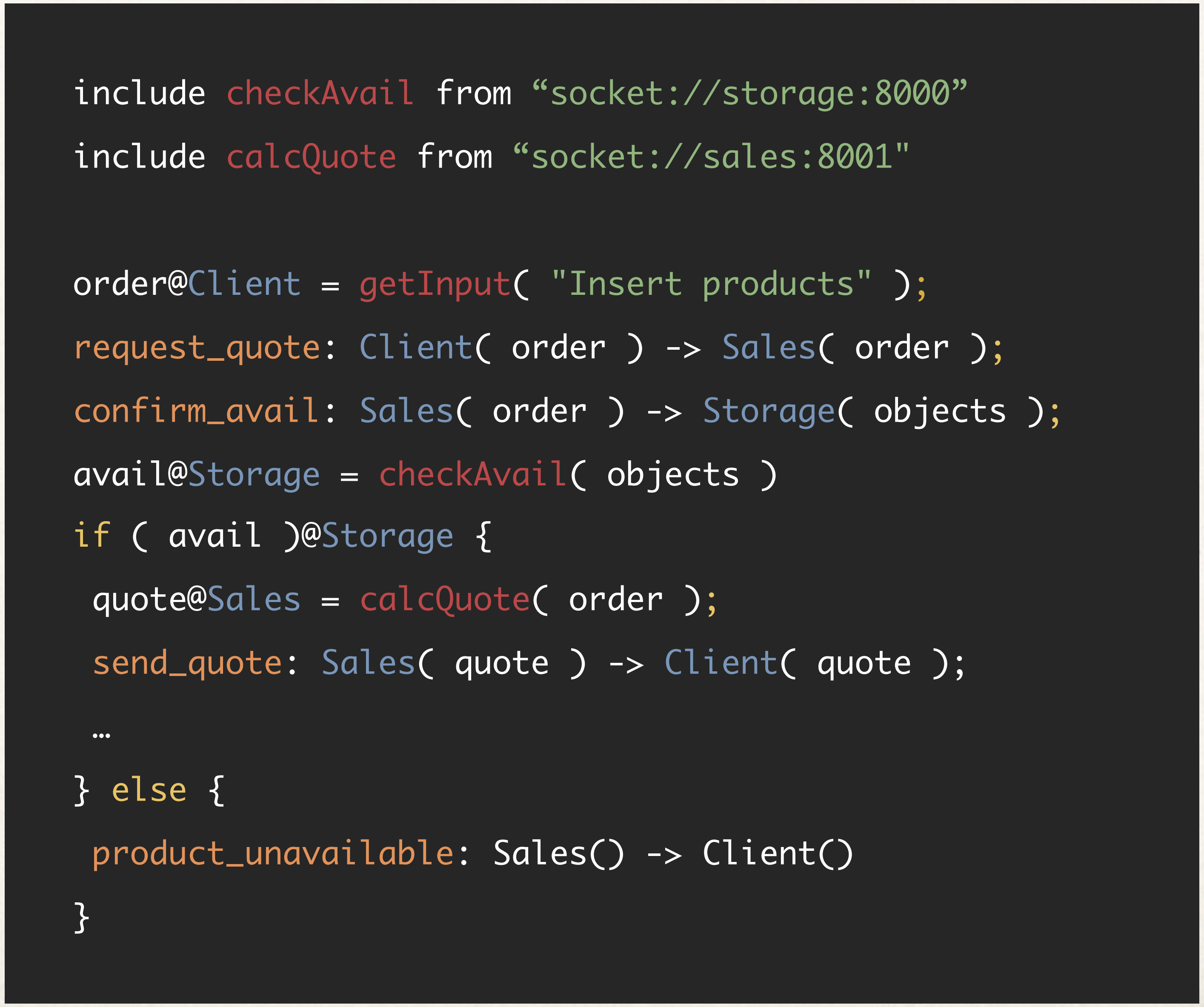
Architectural Vision



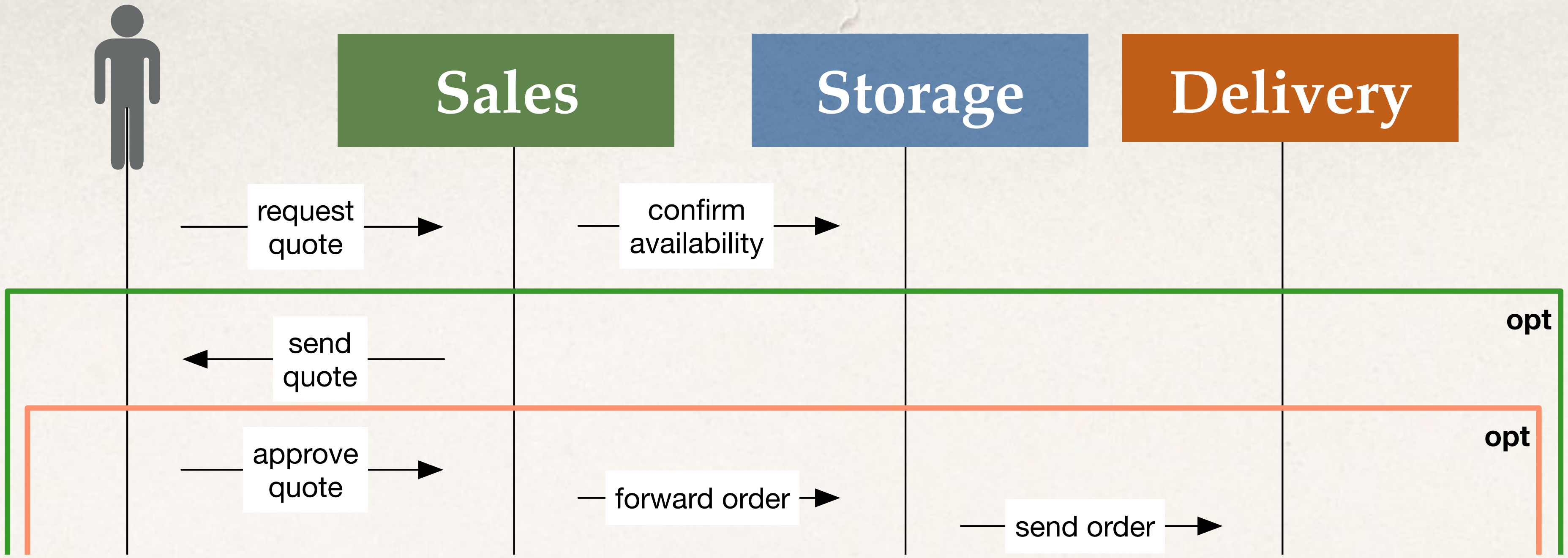
Architectural Vision

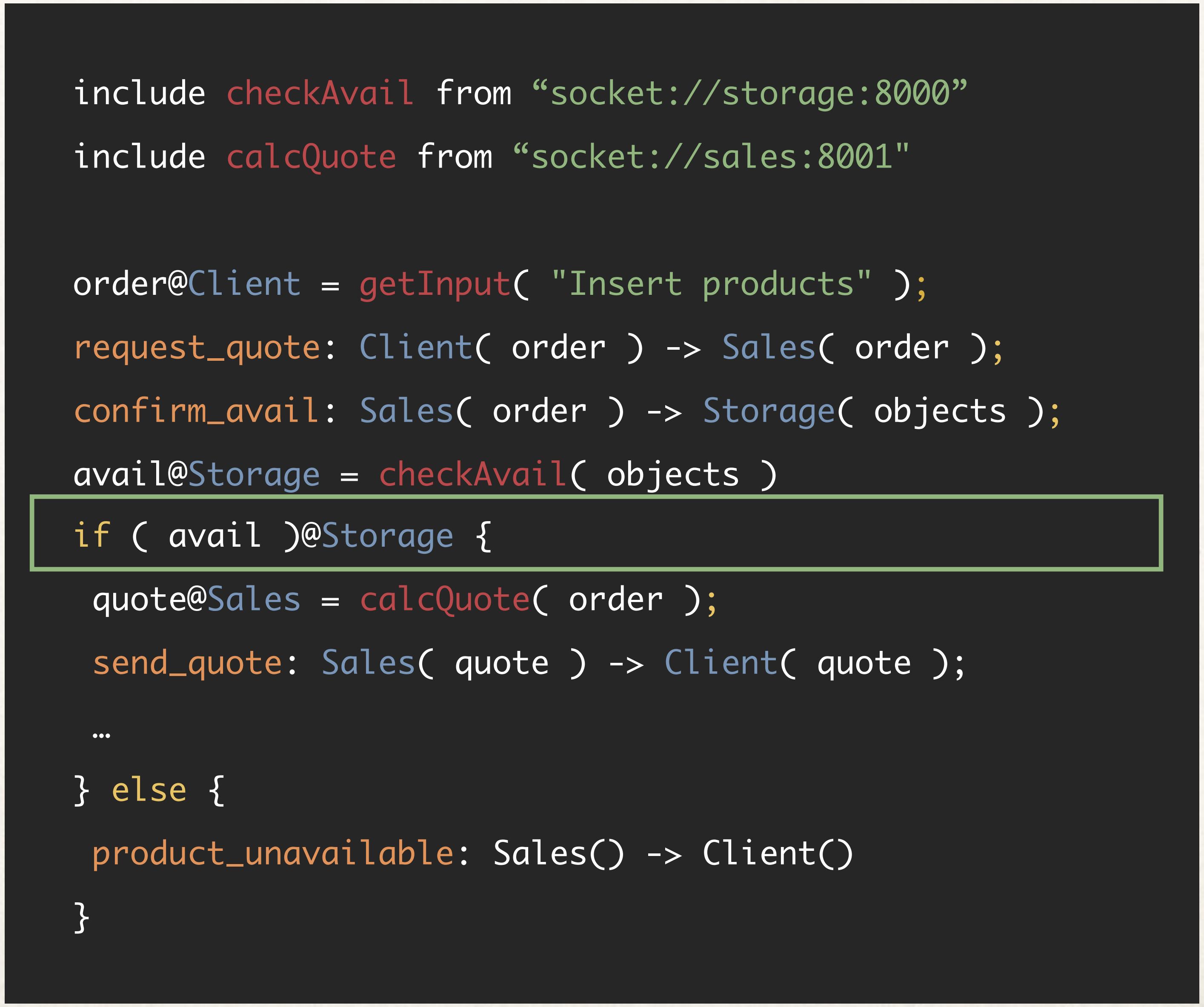




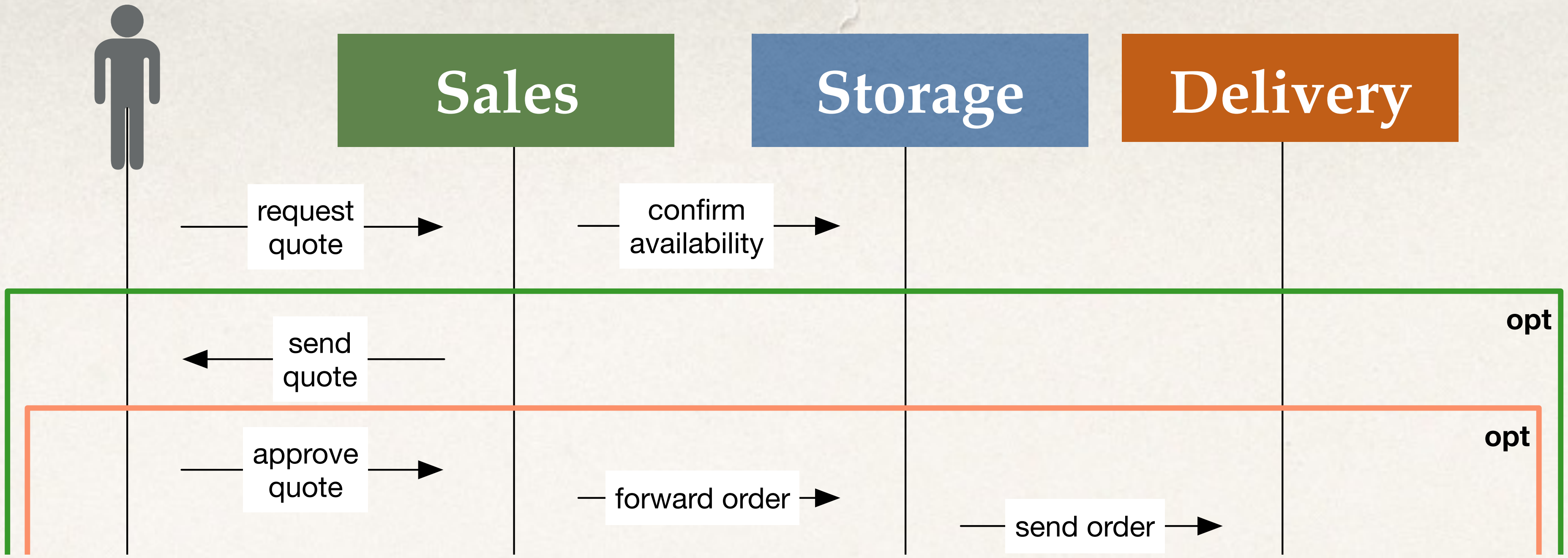


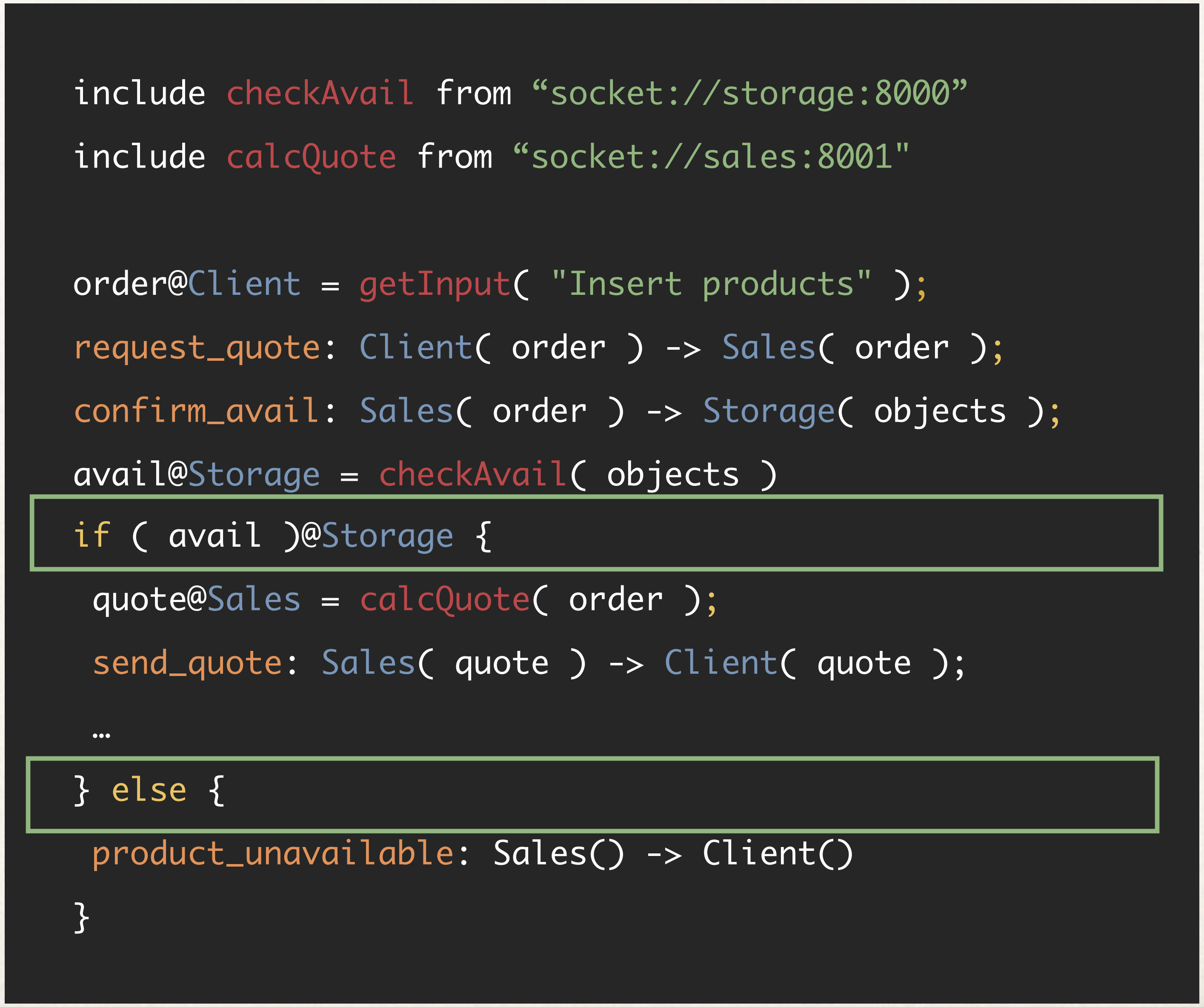
```
include checkAvail from “socket://storage:8000”
include calcQuote from “socket://sales:8001"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
if ( avail )@Storage {
 quote@Sales = calcQuote( order );
 send_quote: Sales( quote ) -> Client( quote );
 …
} else {
 product_unavailable: Sales() -> Client()
}
```





```
include checkAvail from “socket://storage:8000”
include calcQuote from “socket://sales:8001"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
if ( avail )@Storage {
 quote@Sales = calcQuote( order );
 send_quote: Sales( quote ) -> Client( quote );
 …
} else {
 product_unavailable: Sales() -> Client()
}
```





```
include checkAvail from “socket://storage:8000”
include calcQuote from “socket://sales:8001"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
if ( avail )@Storage {
 quote@Sales = calcQuote( order );
 send_quote: Sales( quote ) -> Client( quote );
 …
} else {
 product_unavailable: Sales() -> Client()
}
```

Netflix

Why not peer to peer **choreography**?

Netflix

Why not peer to peer **choreography**?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

Netflix

Why not peer to peer **choreography**?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.

Netflix

Why not peer to peer **choreography**?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

Netflix (cont'd)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

Netflix (cont'd)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.

Netflix (cont'd)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.
- On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).

Netflix (cont'd)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.

- On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).
- Choreographies written in **AIOCJ** are adaptable at runtime!

Netflix (cont'd)

Why not peer to peer choreography?

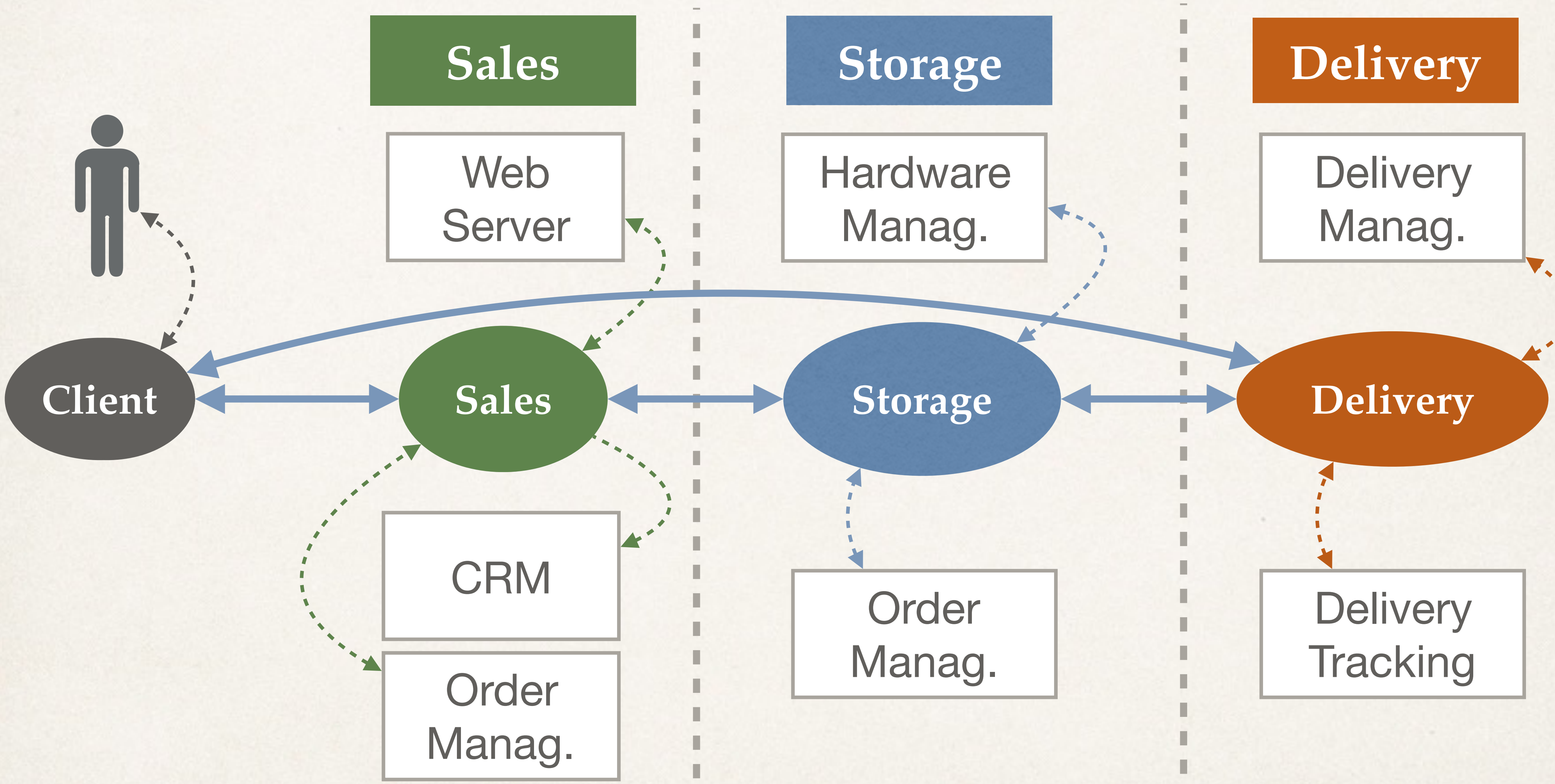
We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

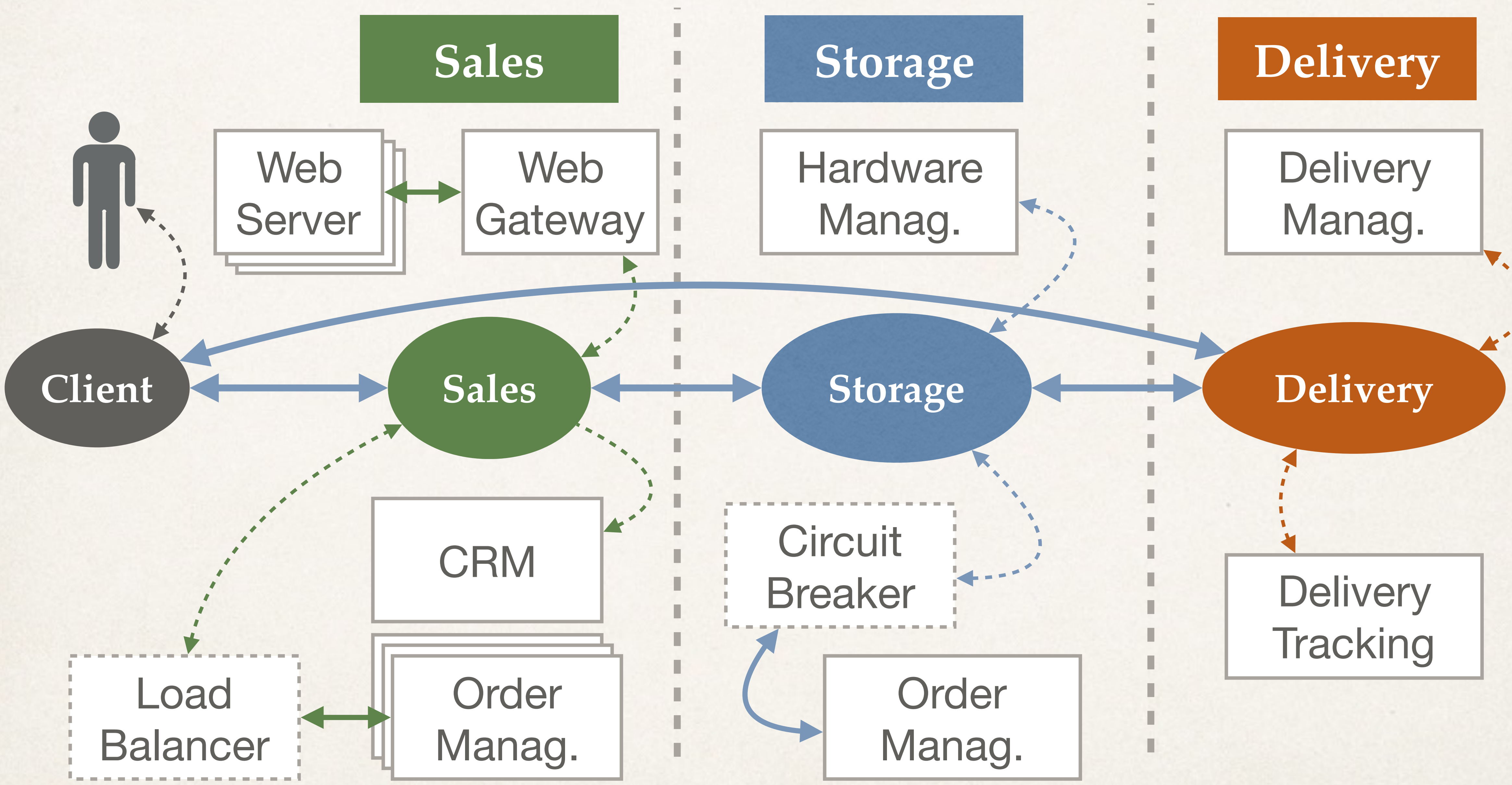
- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.
- On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).
- Choreographies written in **AIOCJ** are adaptable at runtime!

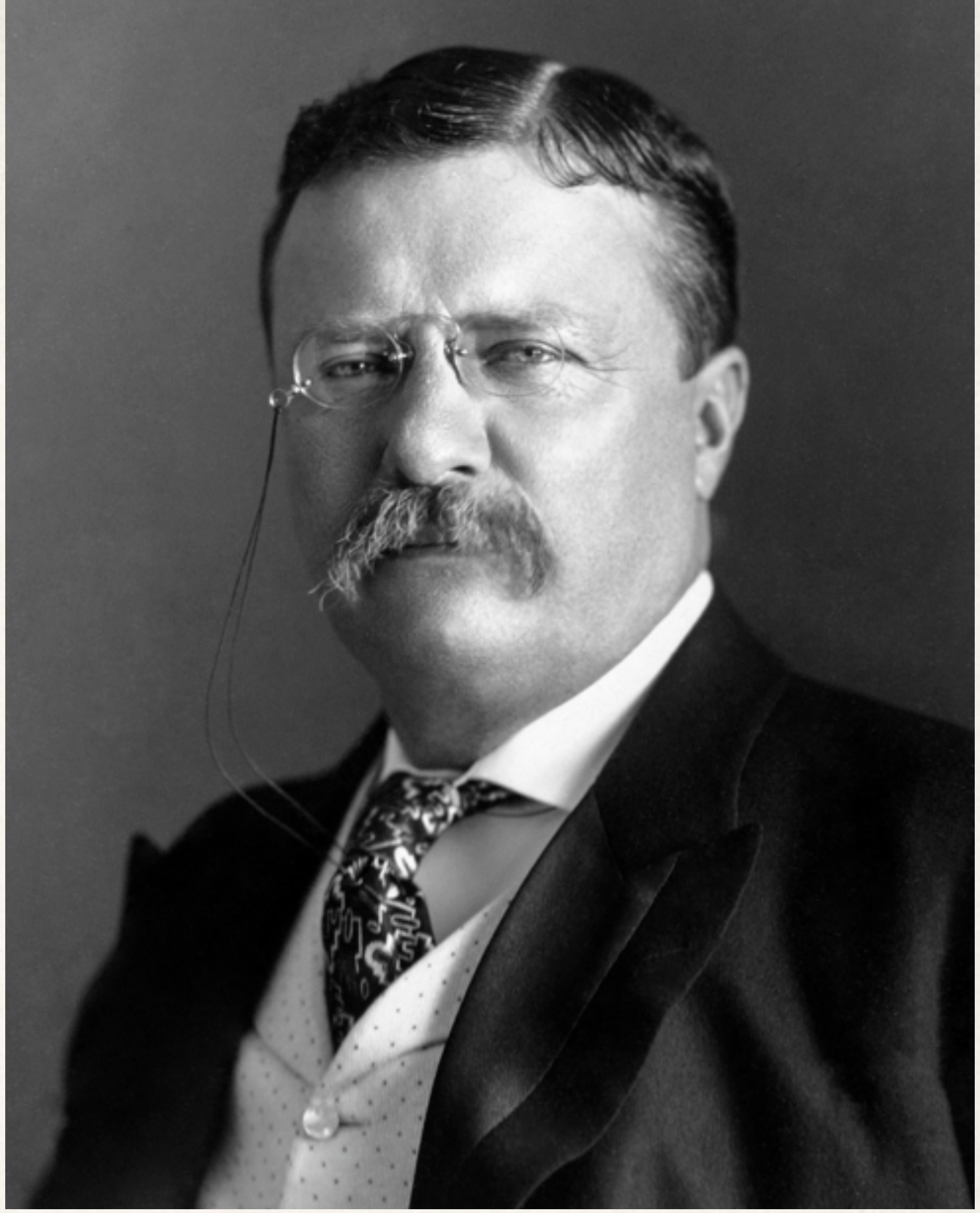
Architectural Vision (Part II)



Architectural Vision (Part III)

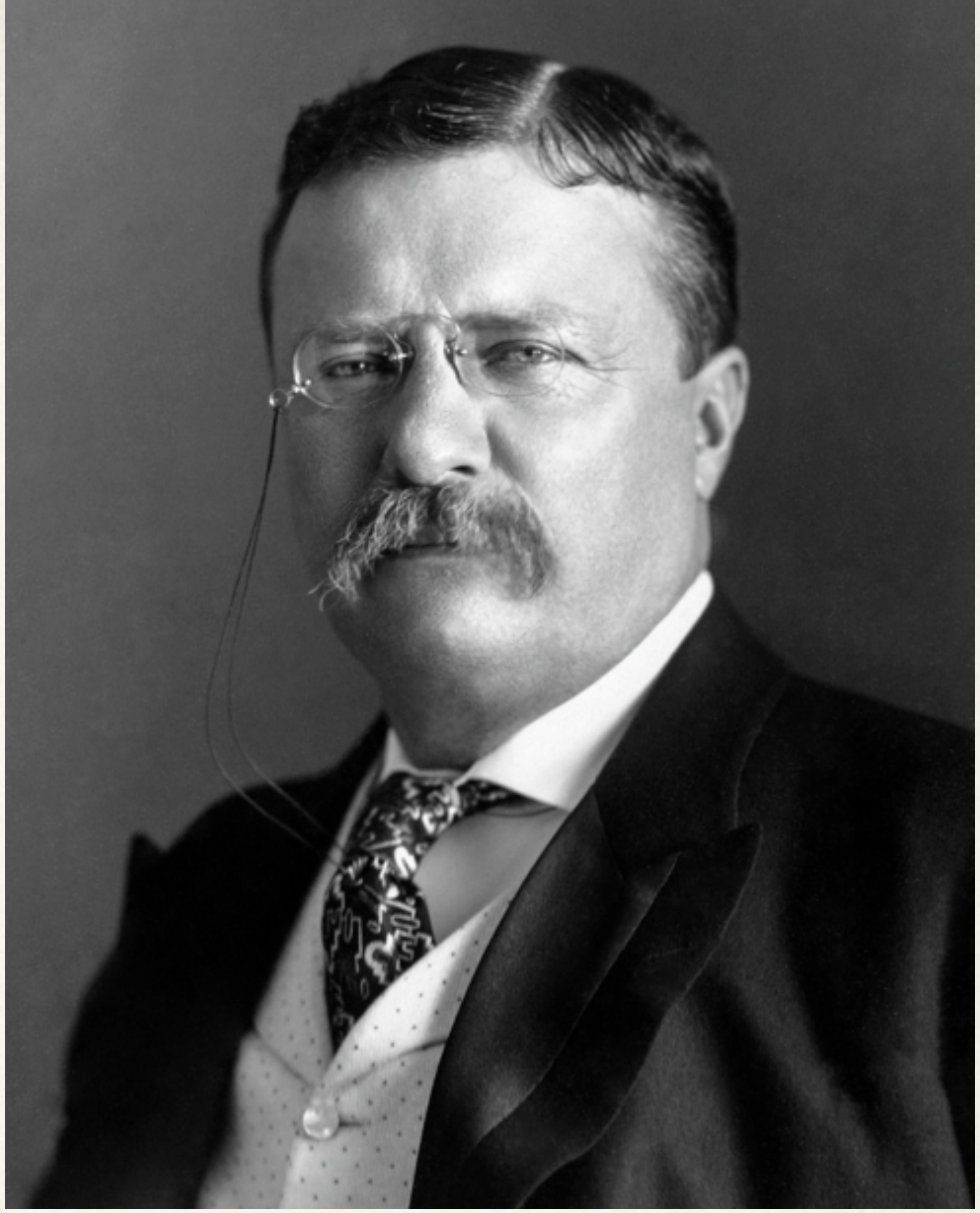


Today's Limits



innovation

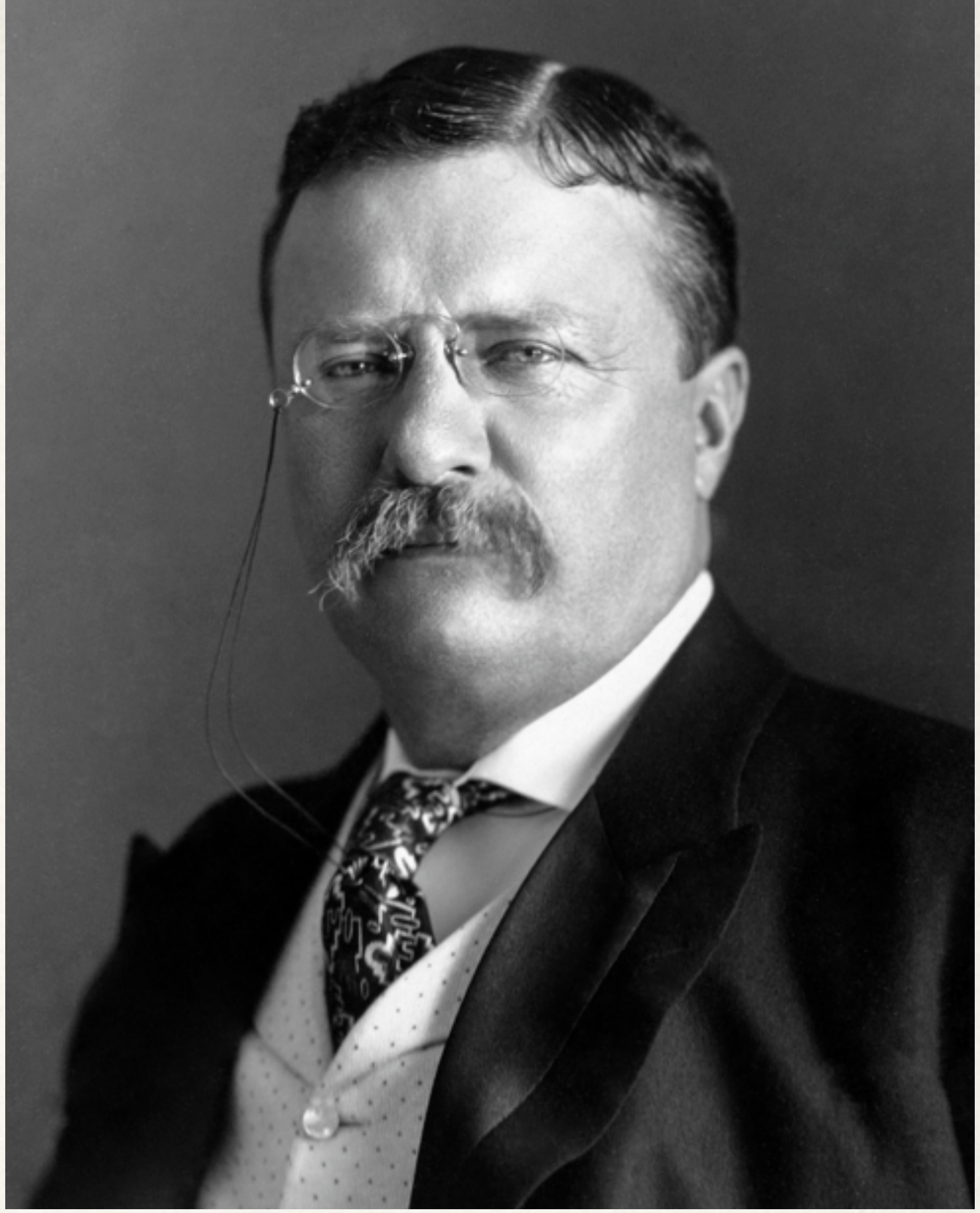
There is no ~~effort~~ without error and shortcoming.



innovation

There is no ~~effort~~ without error and shortcoming.

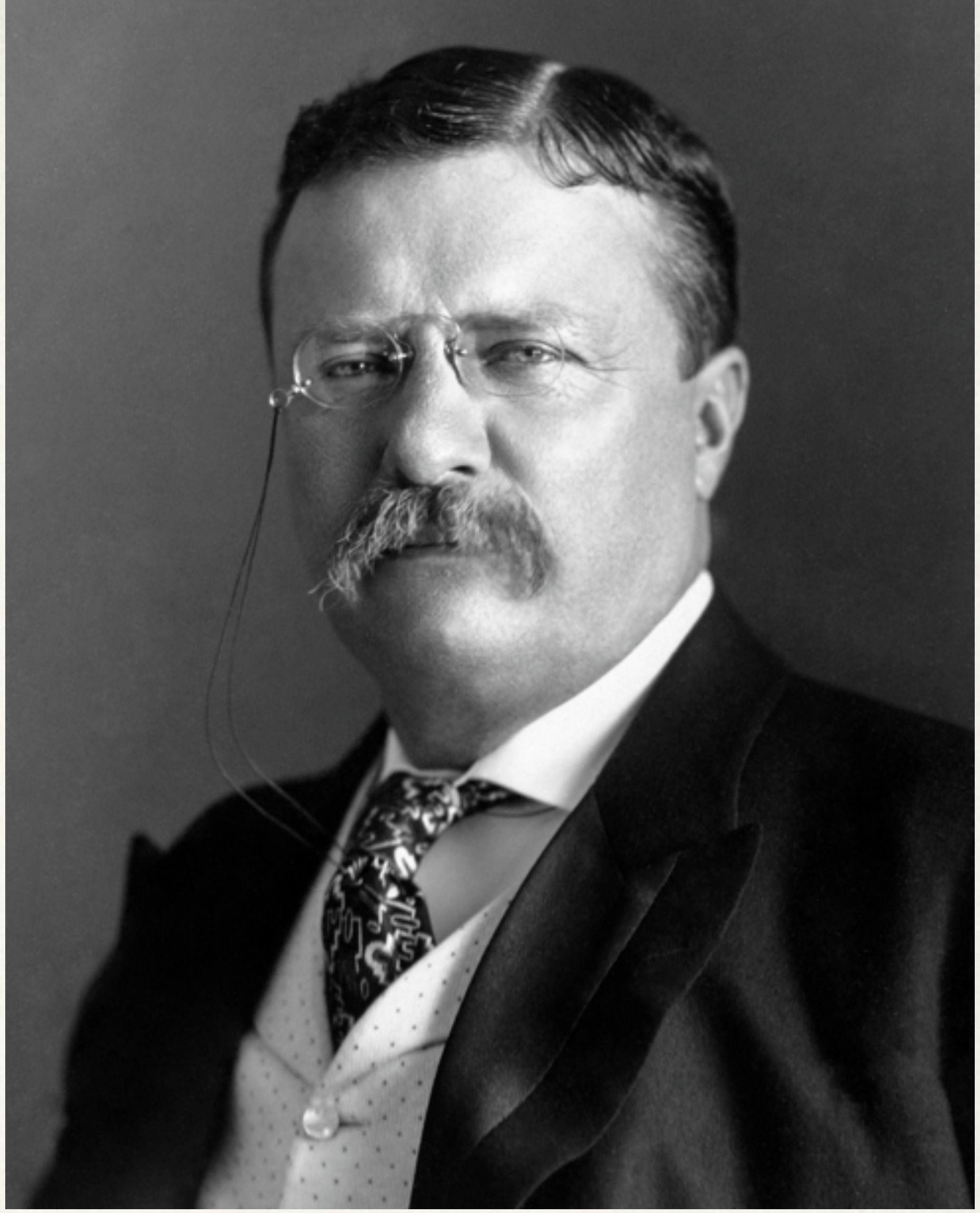
Tomorrow's Standards



innovation

There is no ~~effort~~ without error and shortcoming.

Tomorrow's Standards

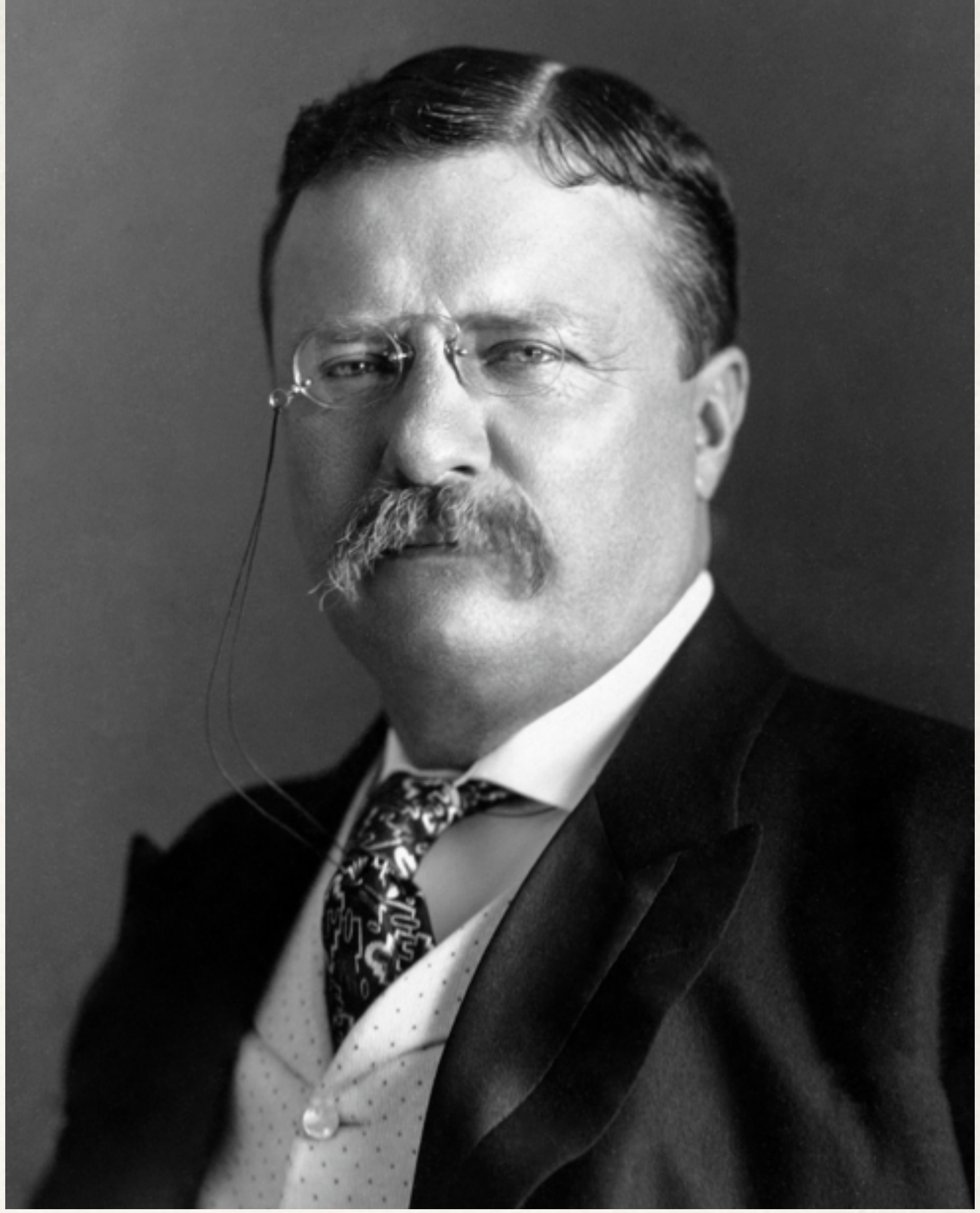


innovation

There is no ~~effort~~ without error and shortcoming.

Tomorrow’s Standards

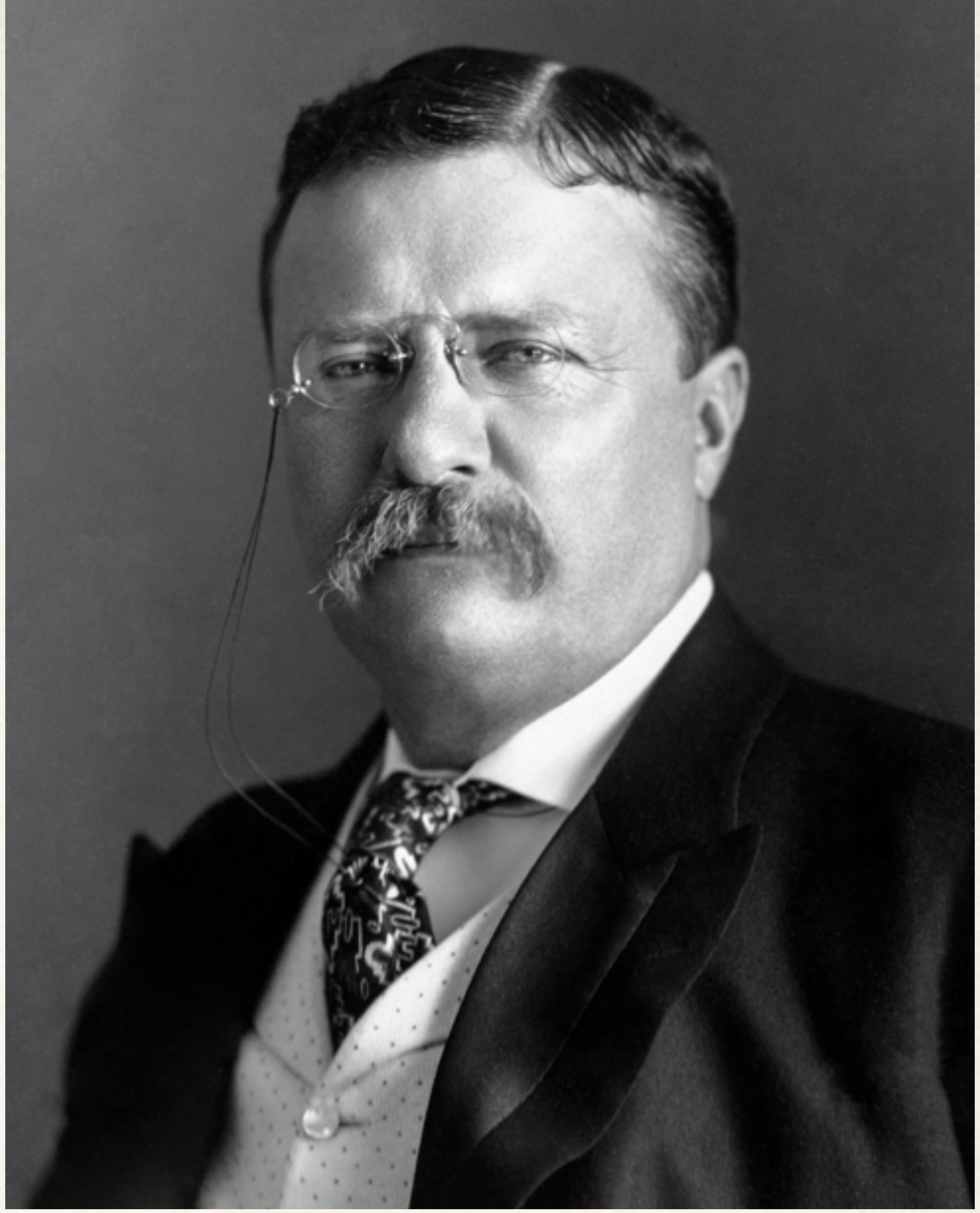
- Distributed programming becomes easier; ☑



innovation
There is no ~~effort~~
without error and
shortcoming.

Tomorrow's Standards

- Distributed programming becomes easier; ☑
- Accountability and formal APIs; ☑

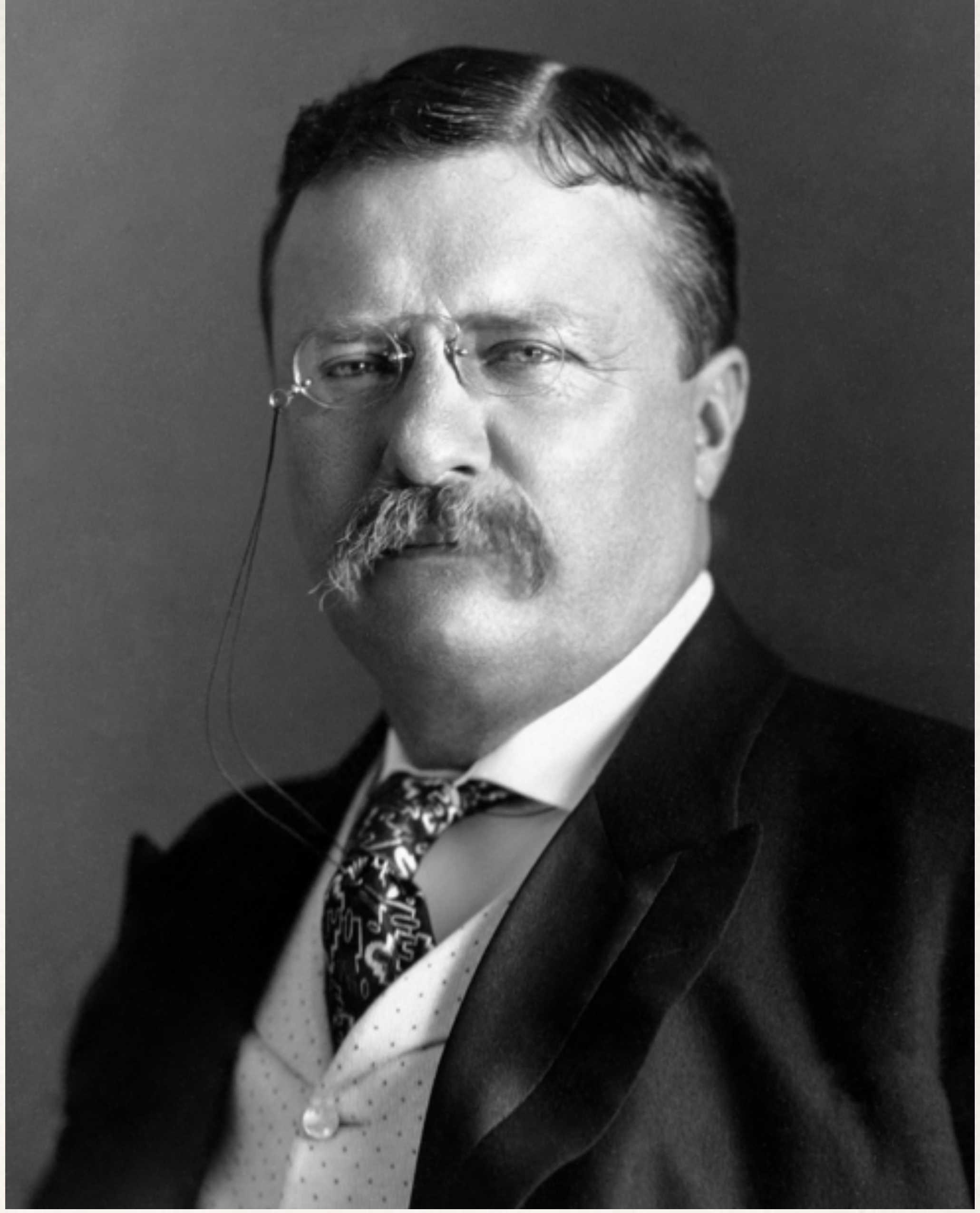


innovation

There is no ~~effort~~
without error and
shortcoming.

Tomorrow's Standards

- Distributed programming becomes easier; ☑
- Accountability and formal APIs; ☑
- Scalable and reliable architectures. ☑



There is no ~~effort~~ innovation without error and shortcoming.

Thanks for the attention

```
Questions: Saverio( ? ) -> MoM2016( ! )
```