Complex business logic

Ubiquitous language

Domain-driven Design

Domain modeling

DDD & Microservices

At last, some boundaries!

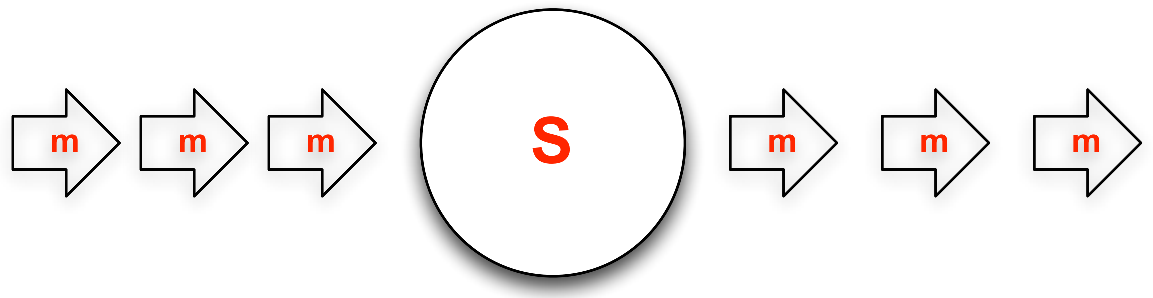
Eric Evans
@ericevans0
domainlanguage.com

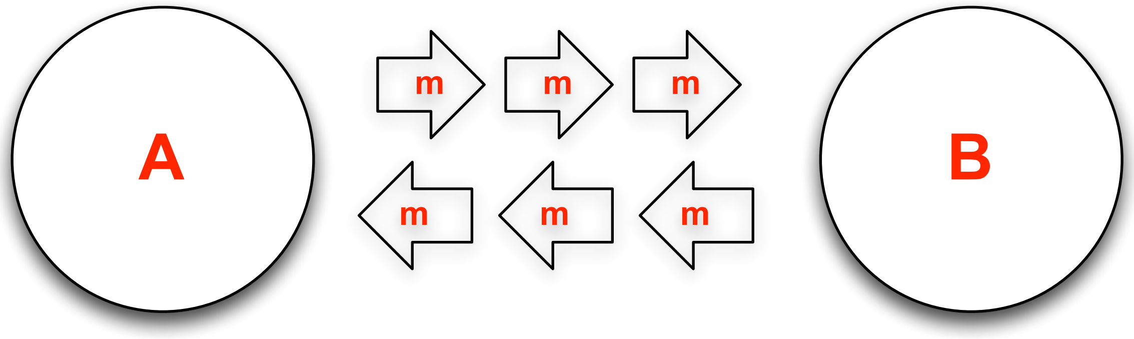
You say ‘bandwagon’ like it’s a bad thing!

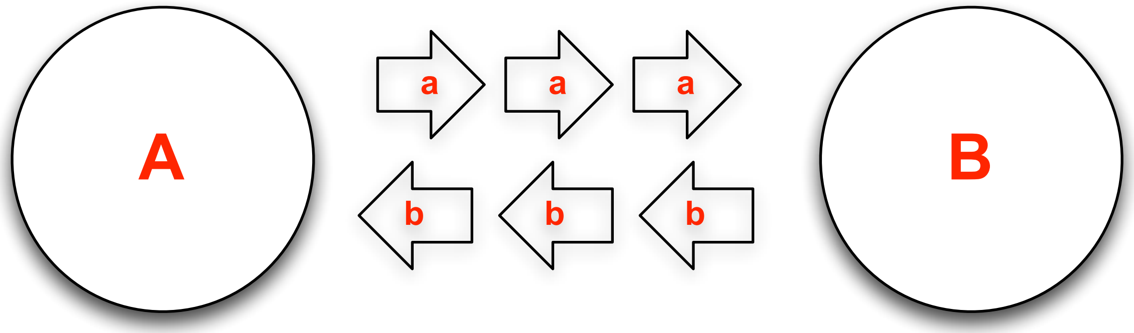
Why do I like microservices?

- Autonomous teams with isolated implementation.
- Acknowledge the rough and tumble of enterprises.
- Cattle not pets.
- A philosophical break from the past — gives us a chance to shake assumptions.
- But! Different people mean different things.

Services and Messages



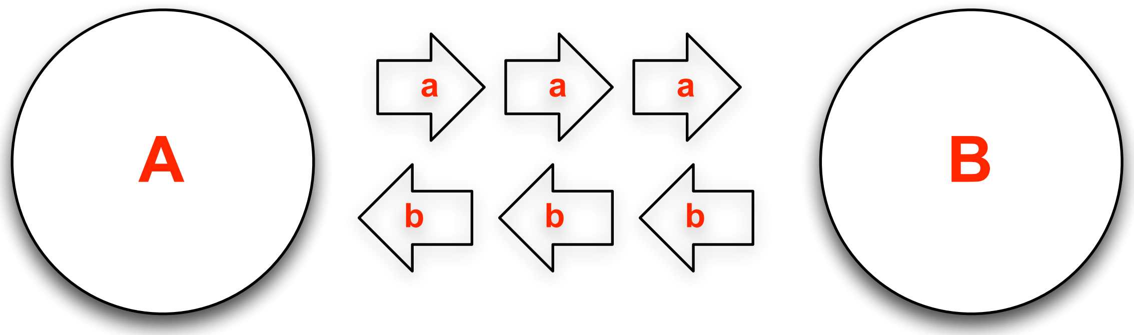

A
m
m
m
m
m
m
B



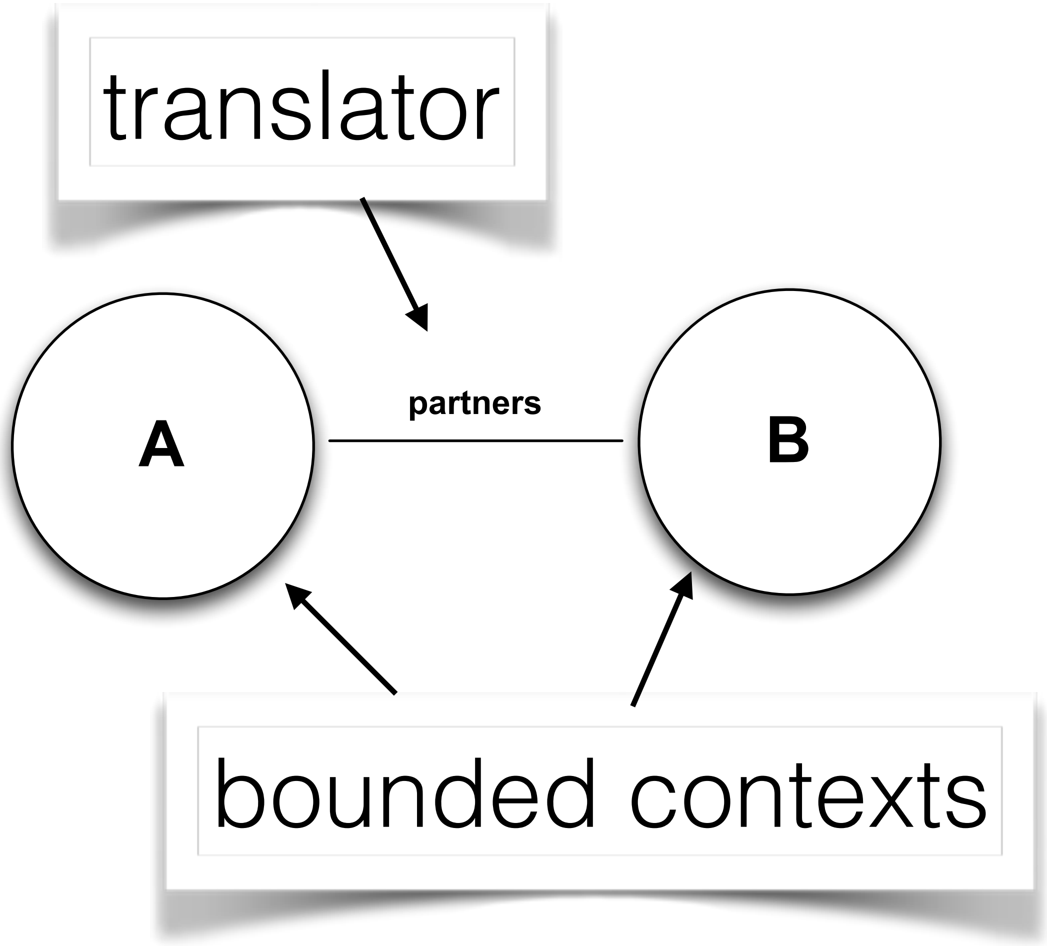
How do they understand the messages?

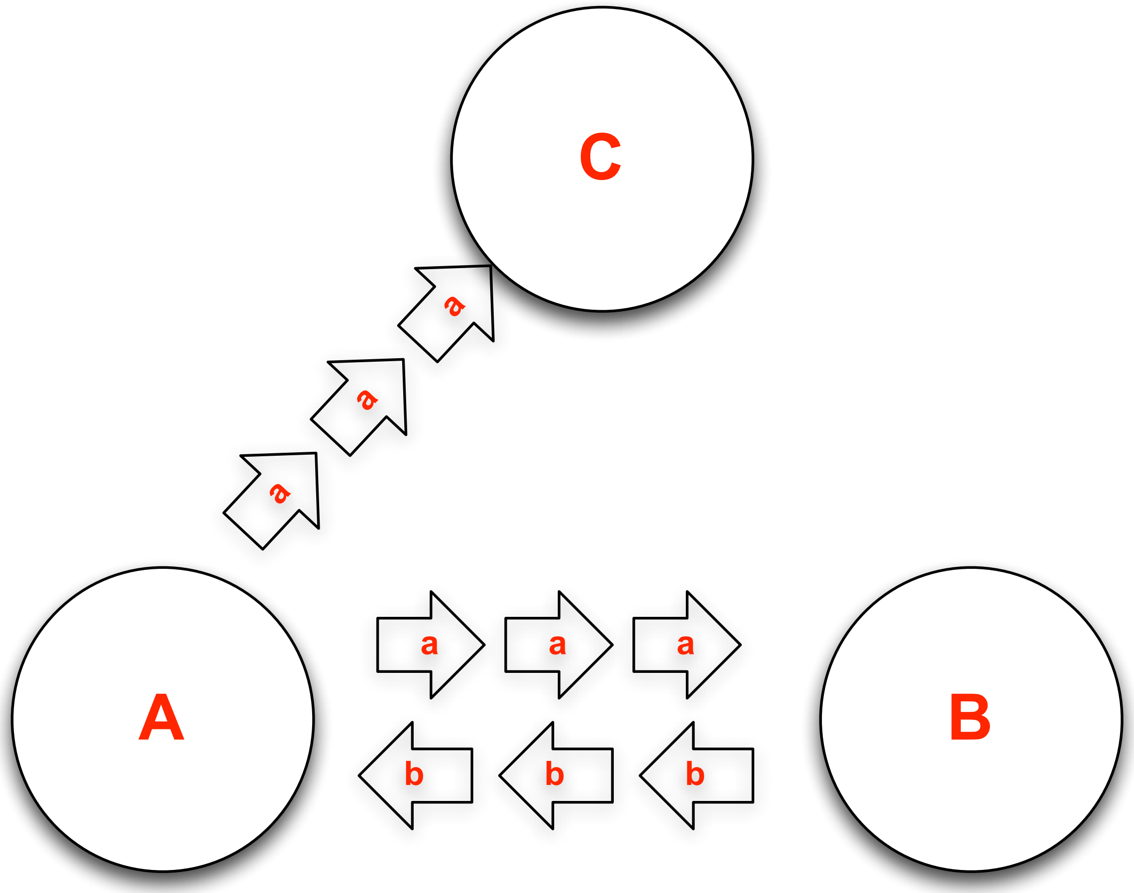
Bounded Context

- context	The setting in which a word or statement appears that determines its meaning
- bounded context The conditions under which a particular model is defined and applicable.

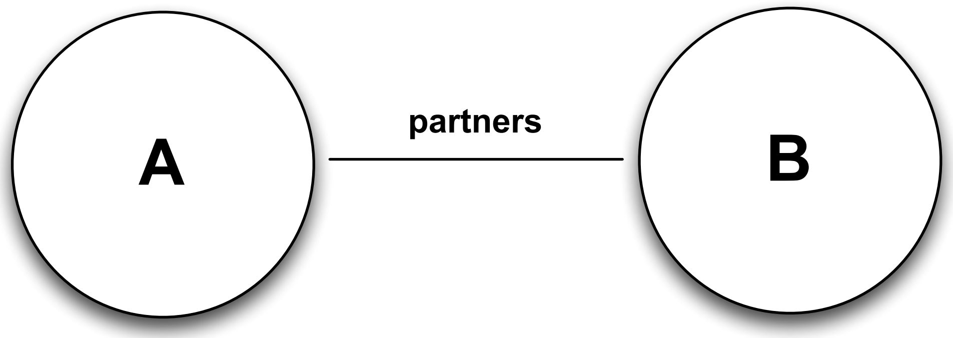


Context Map

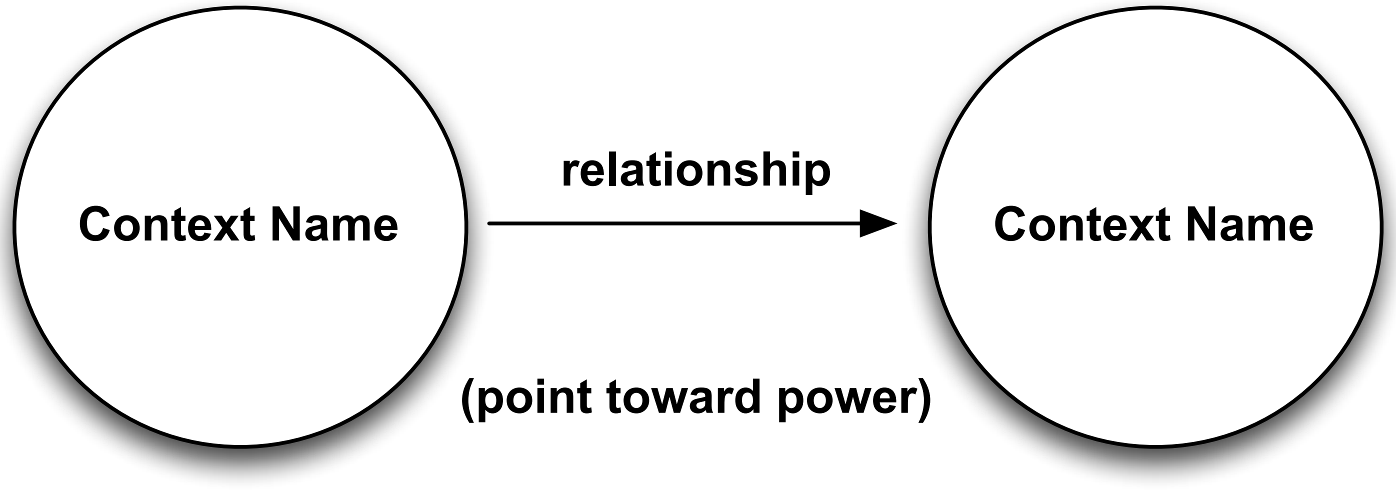


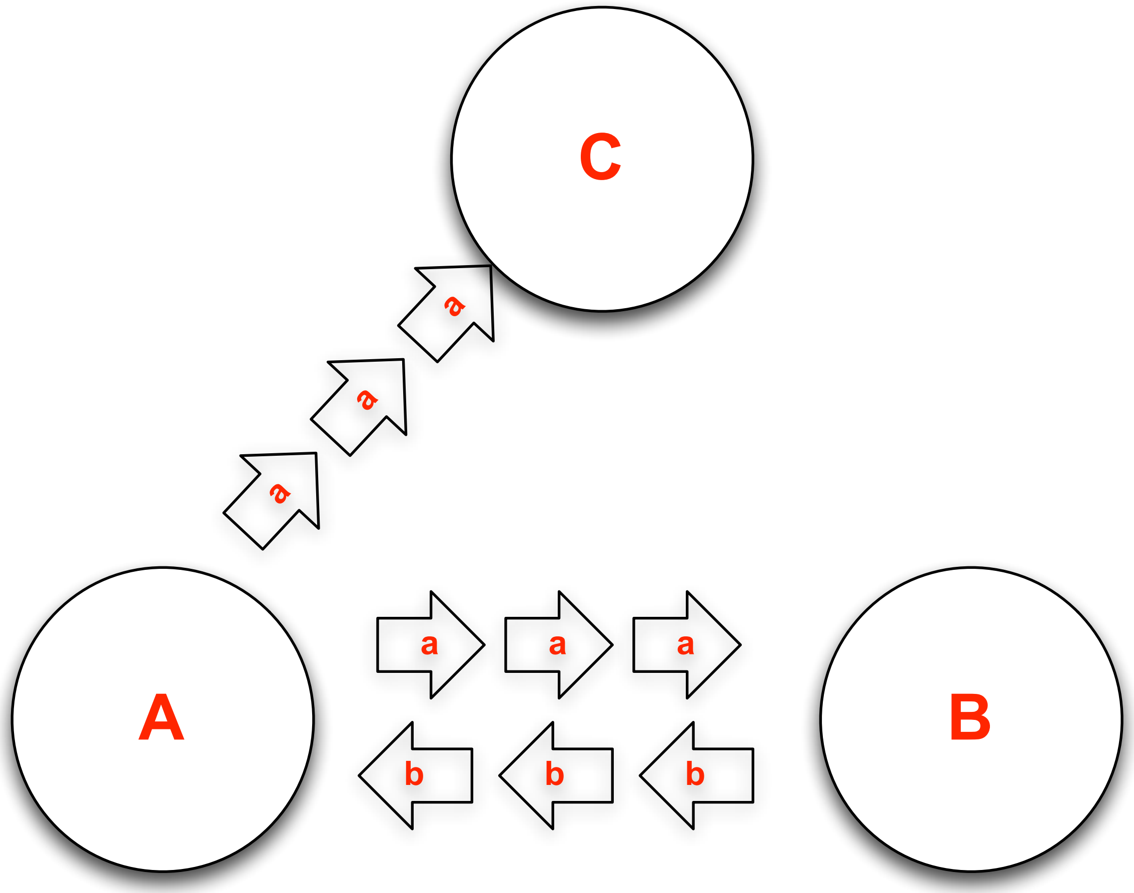


Context Map

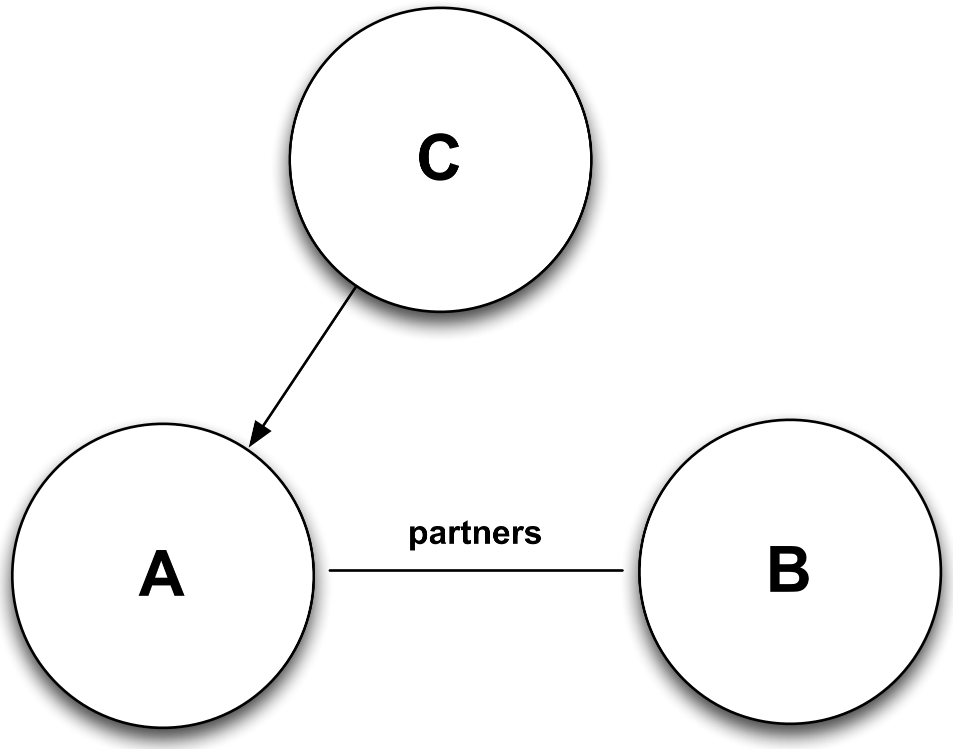


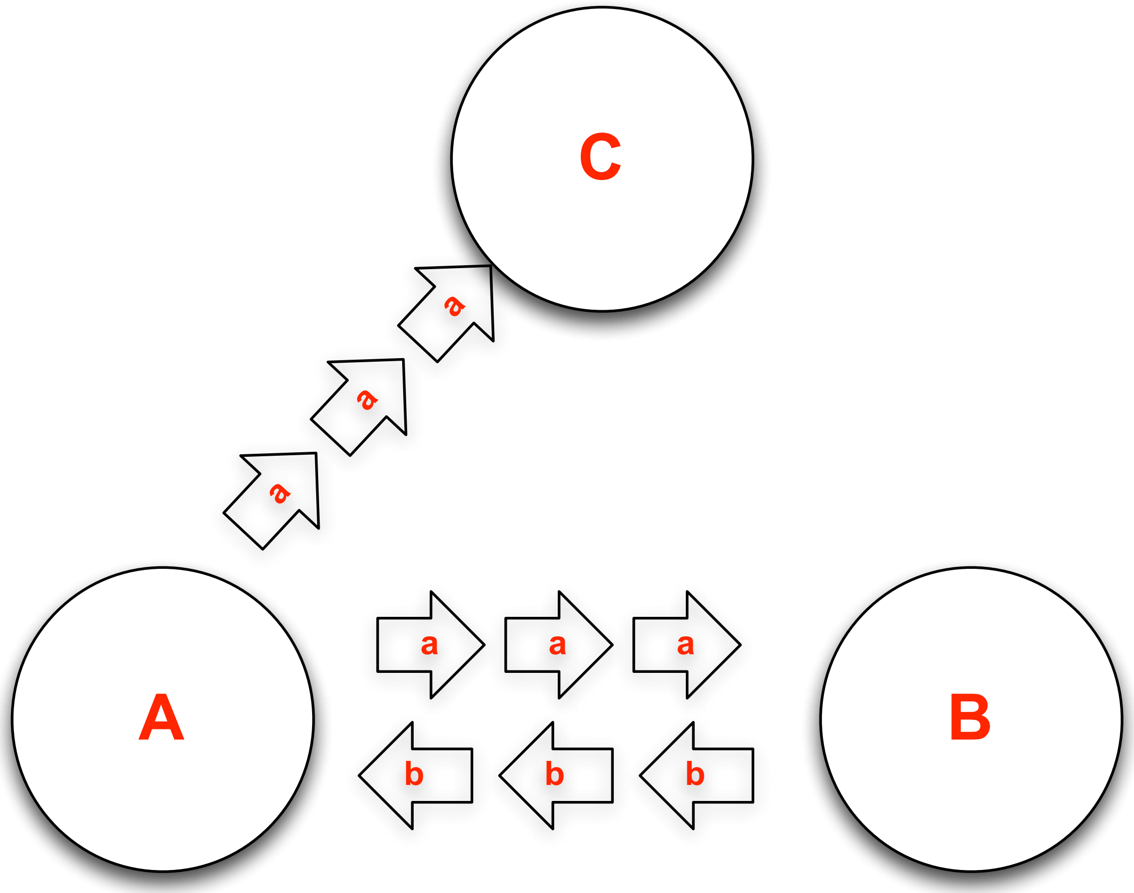
Asymmetrical Relationships



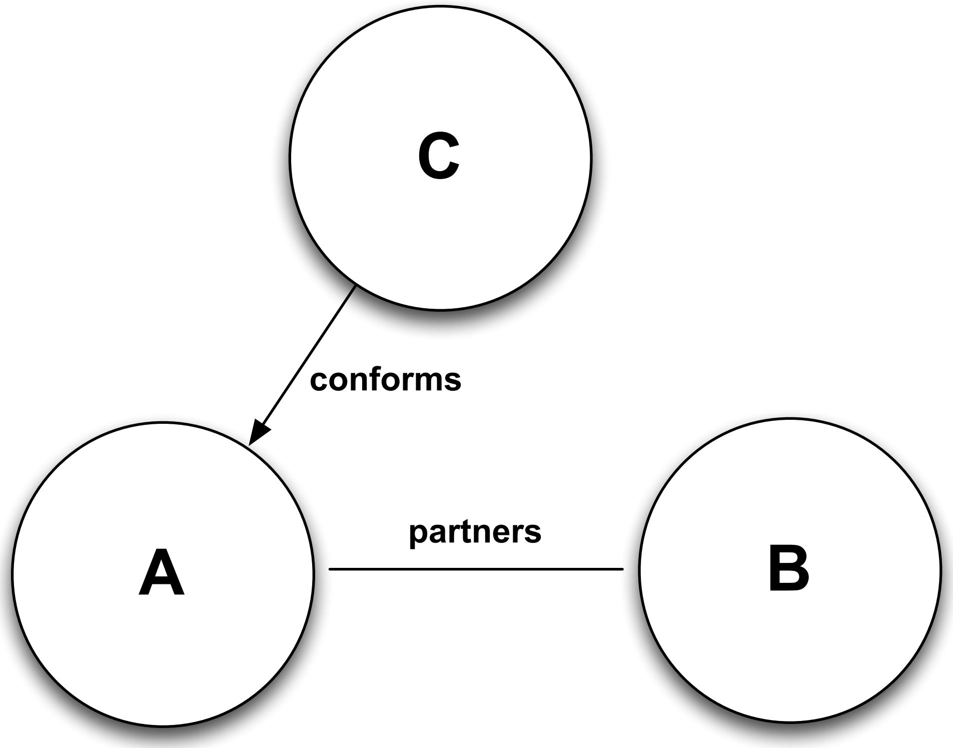


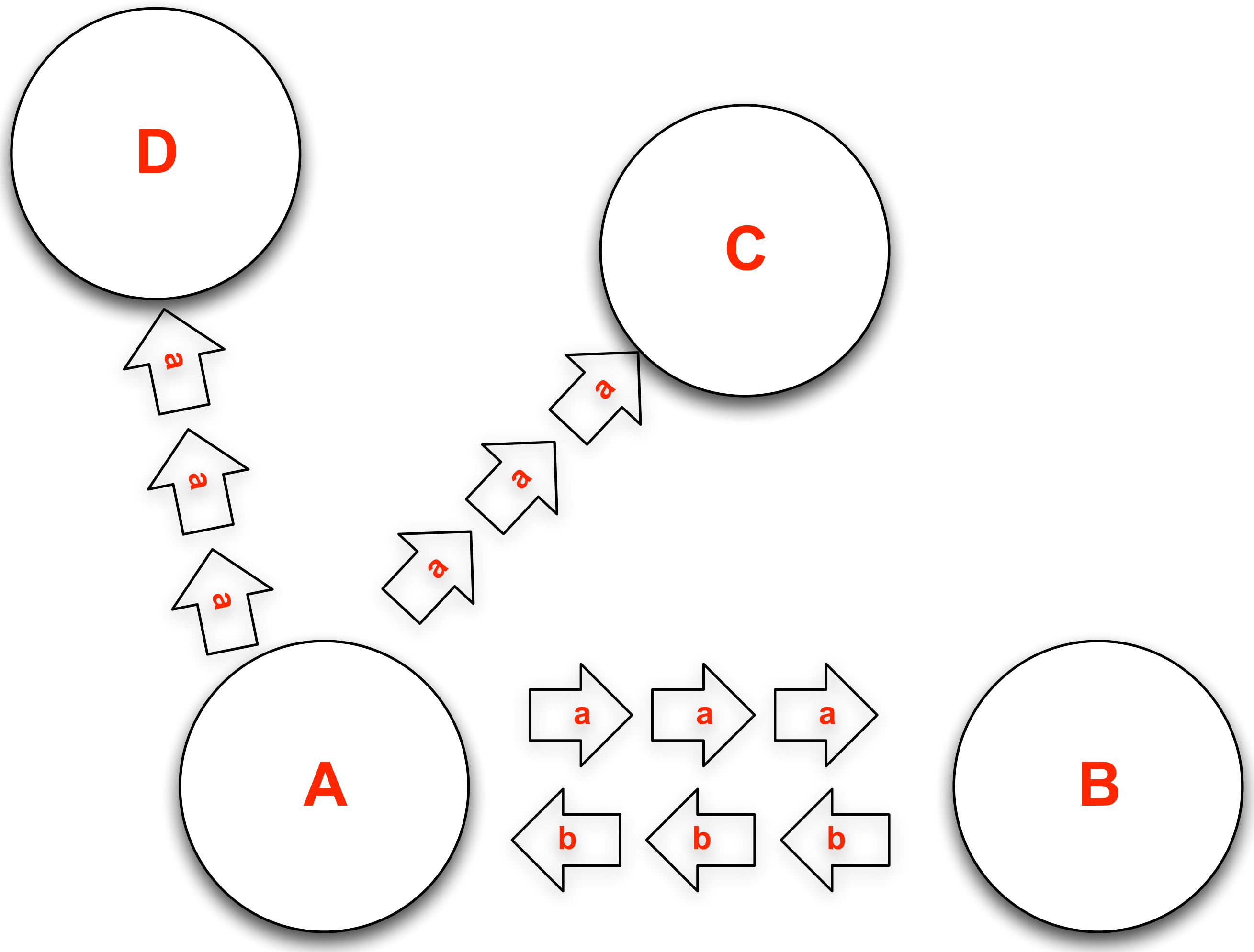
Context Map



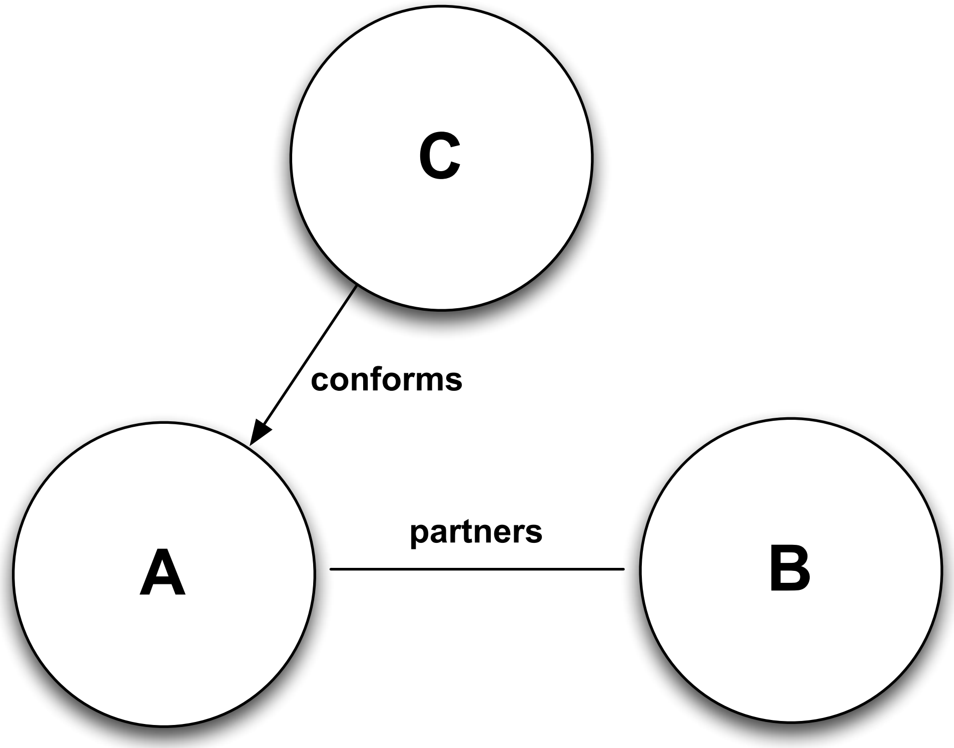


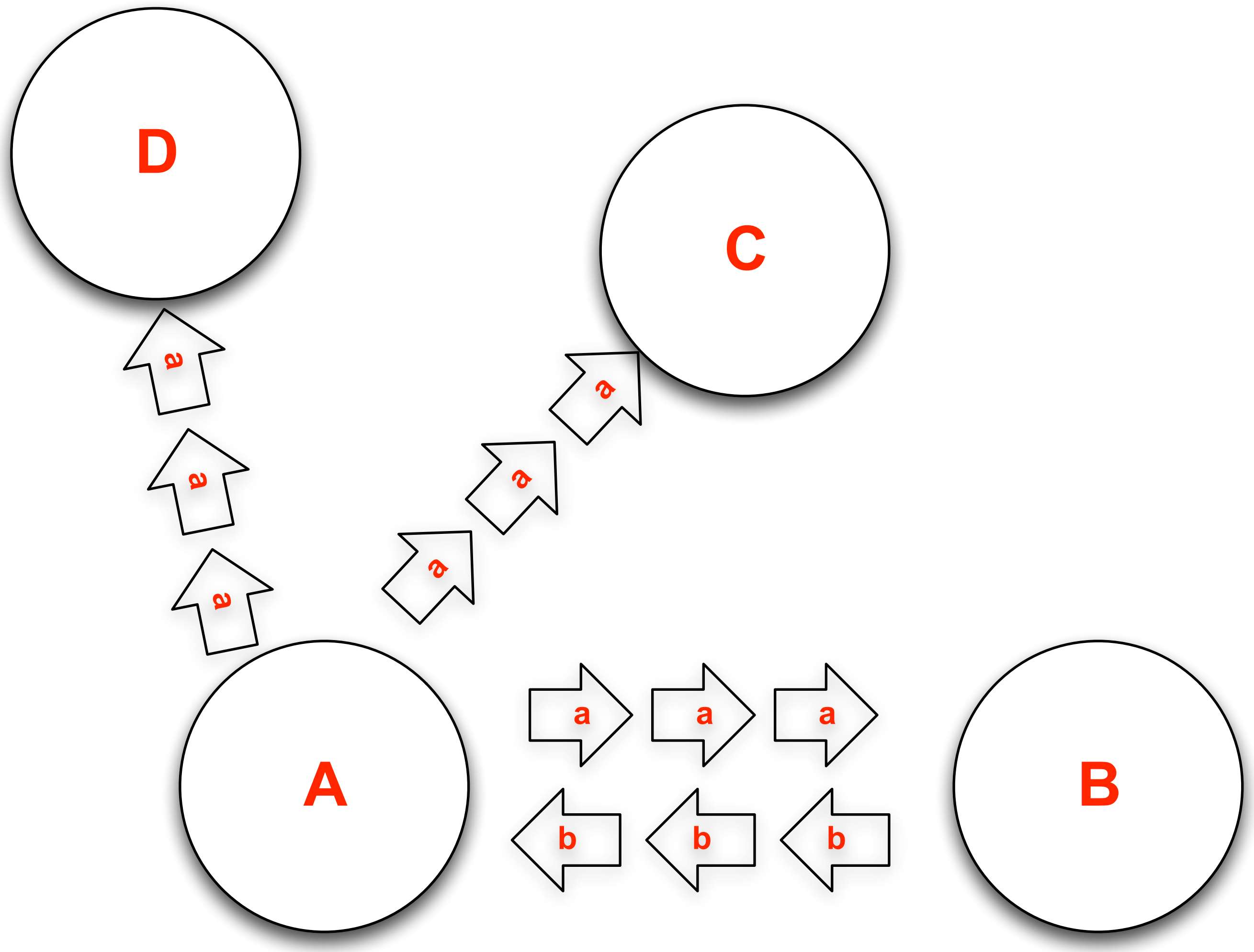
Context Map



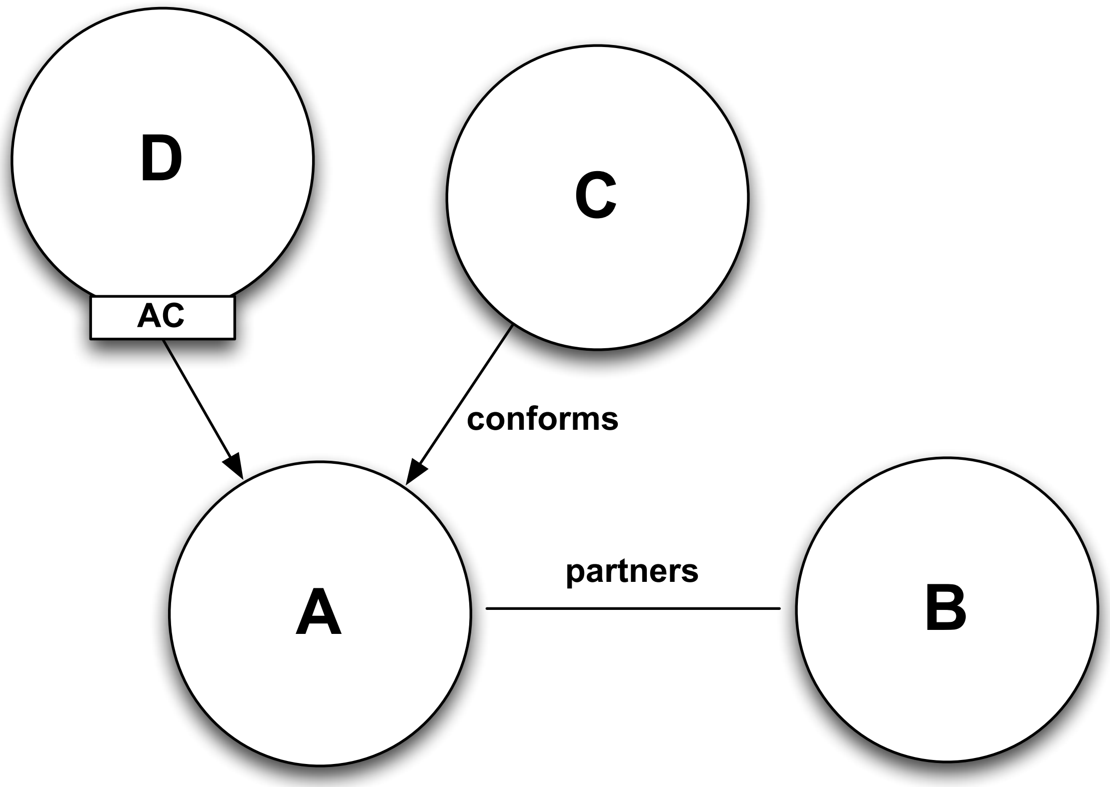


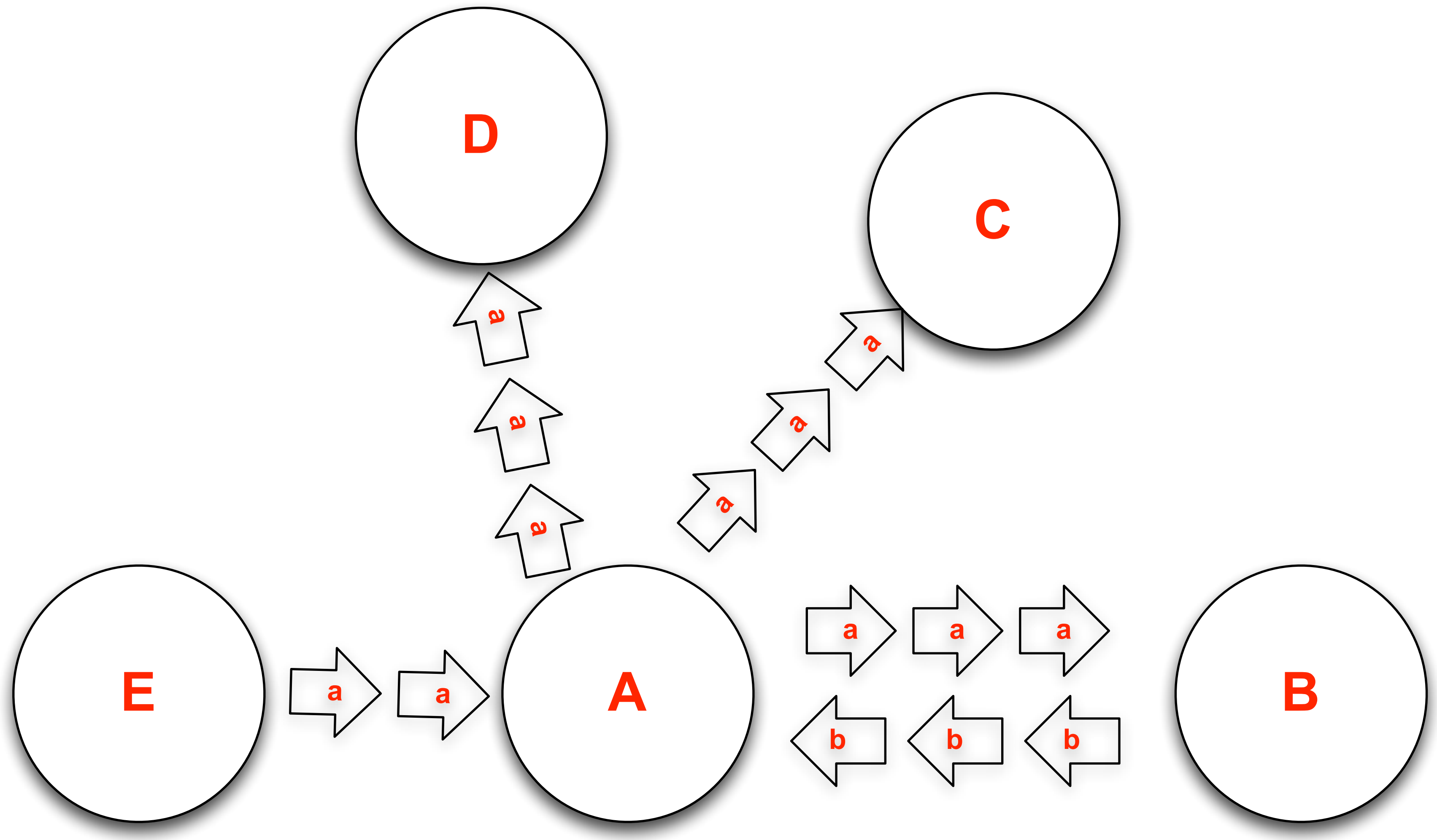
Context Map



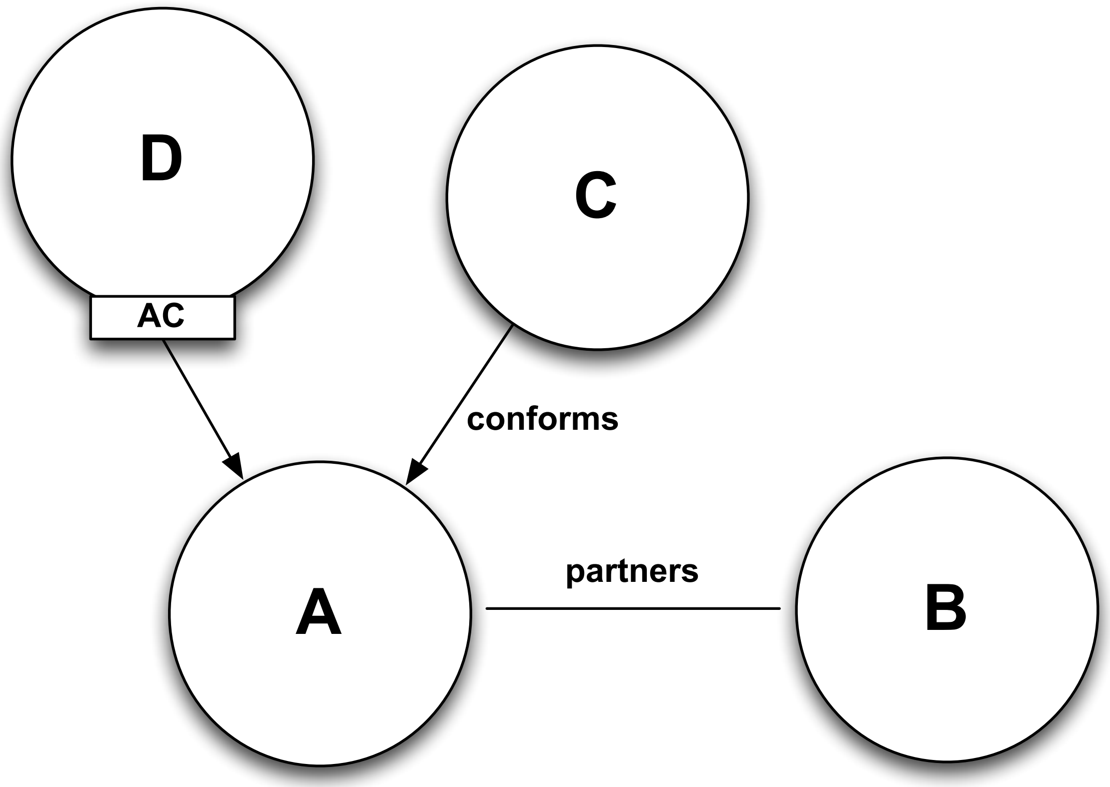


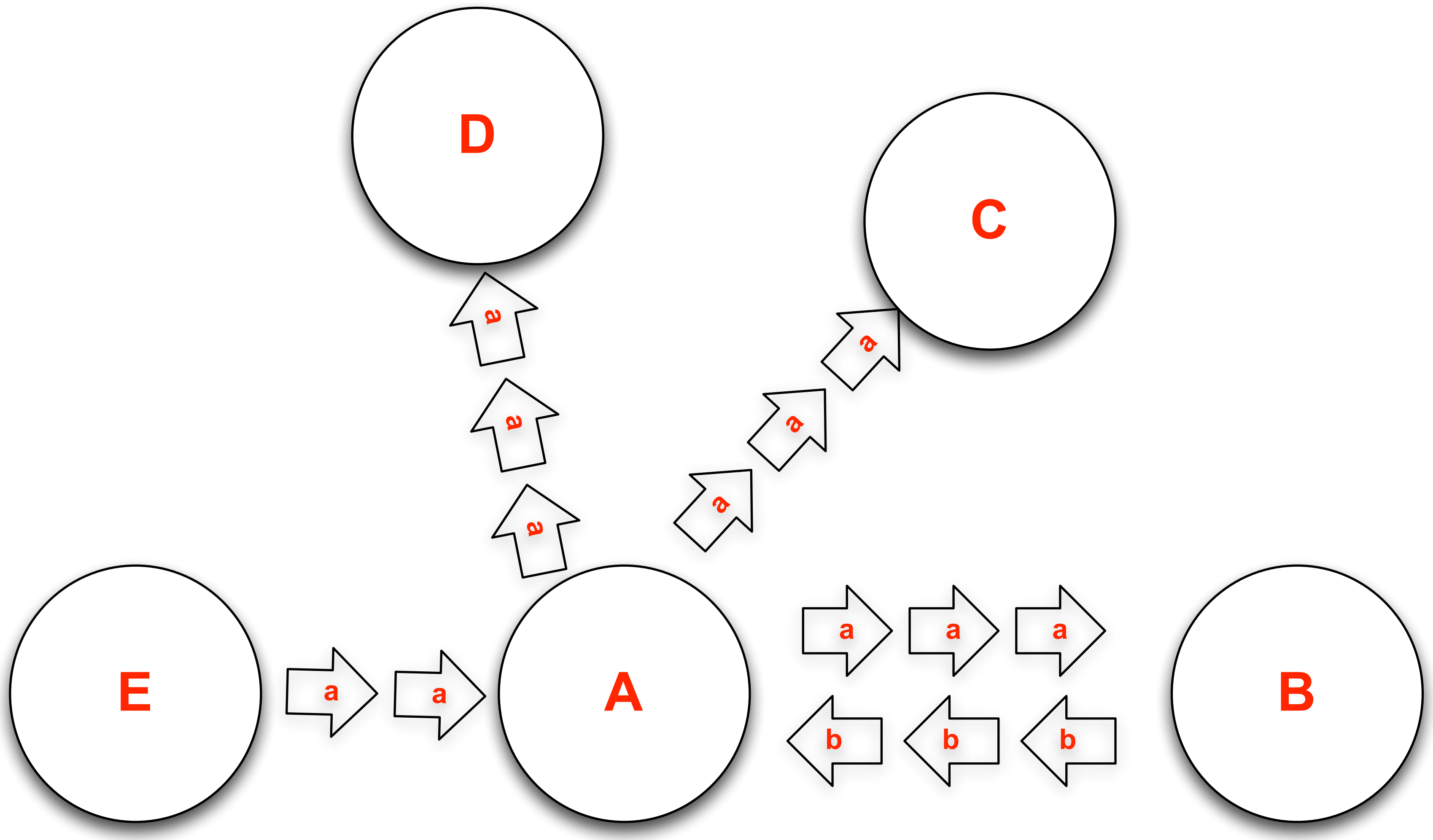
Context Map





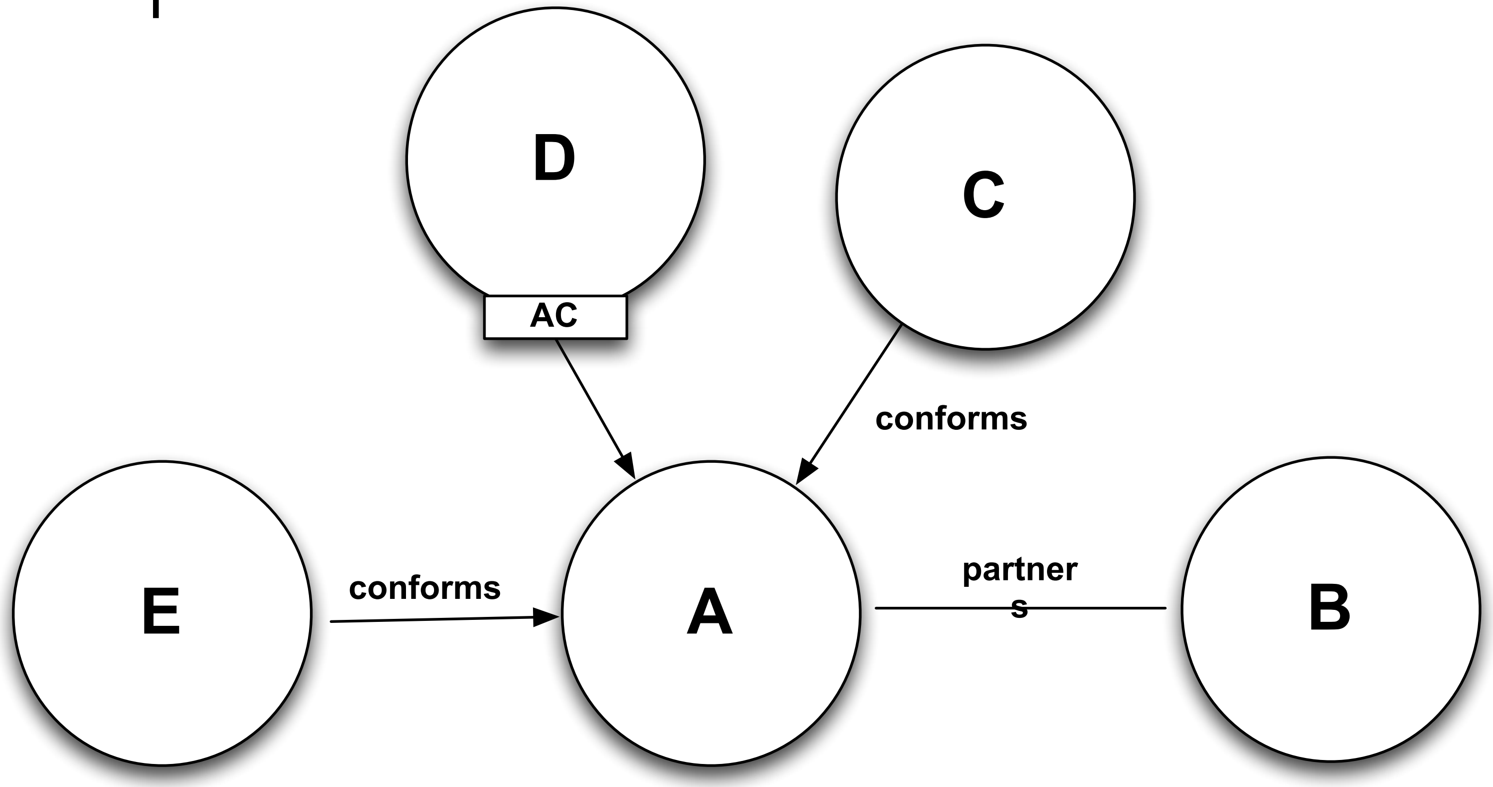
Context Map



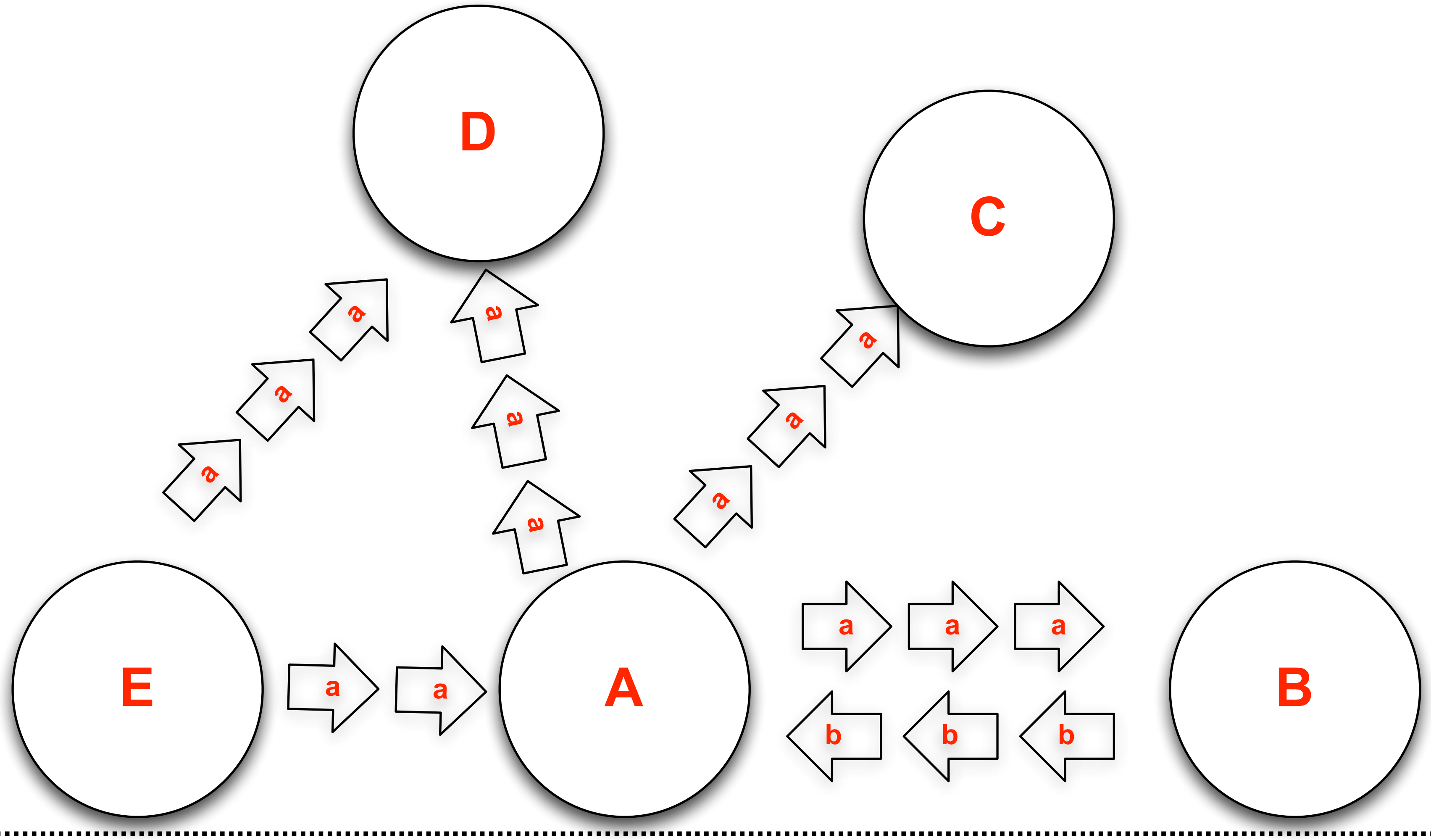


D
C
E
A
B
a
a
a
a
a
a
a
a
a
a
a
b
b
b

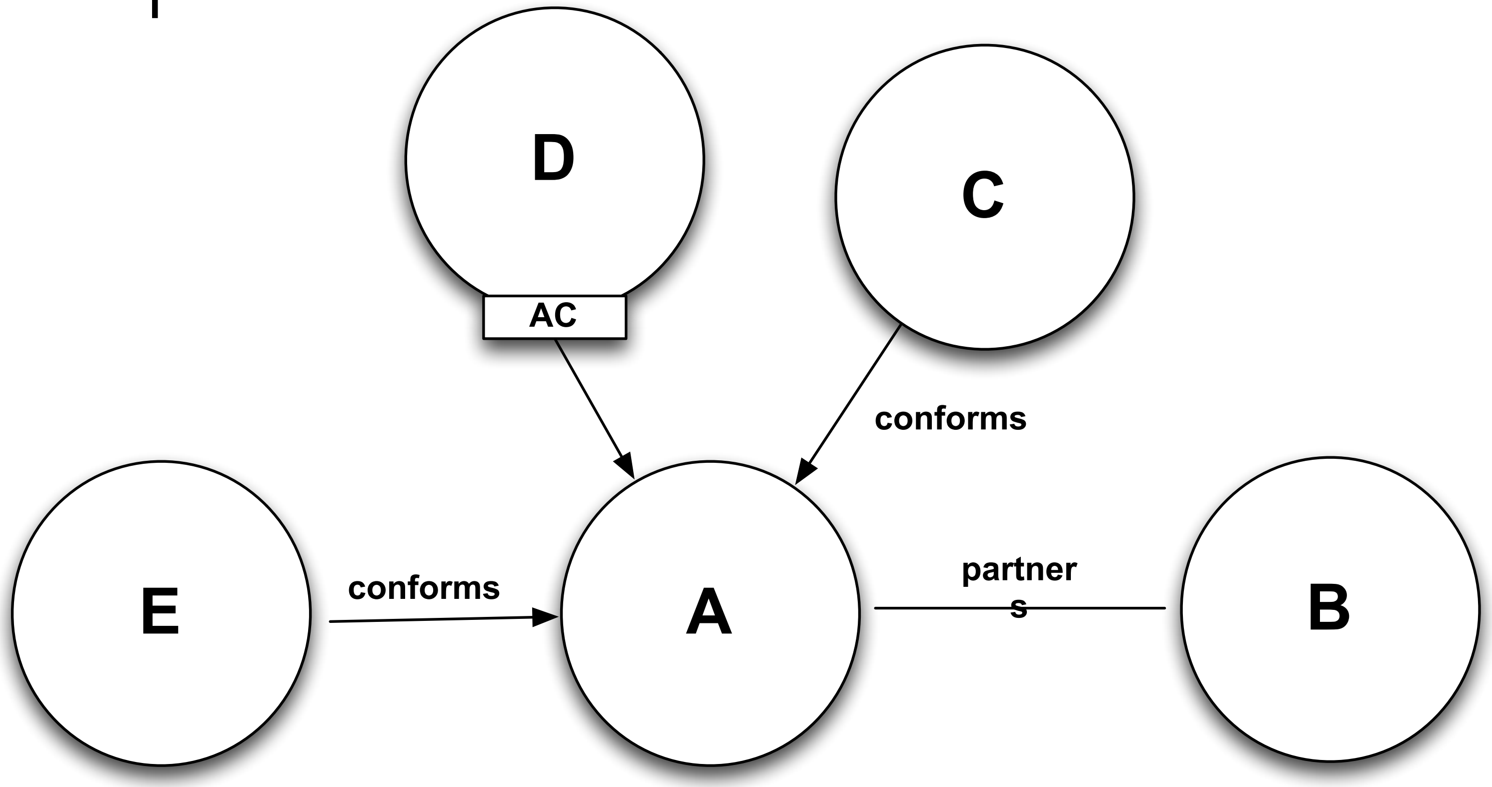
Context Map

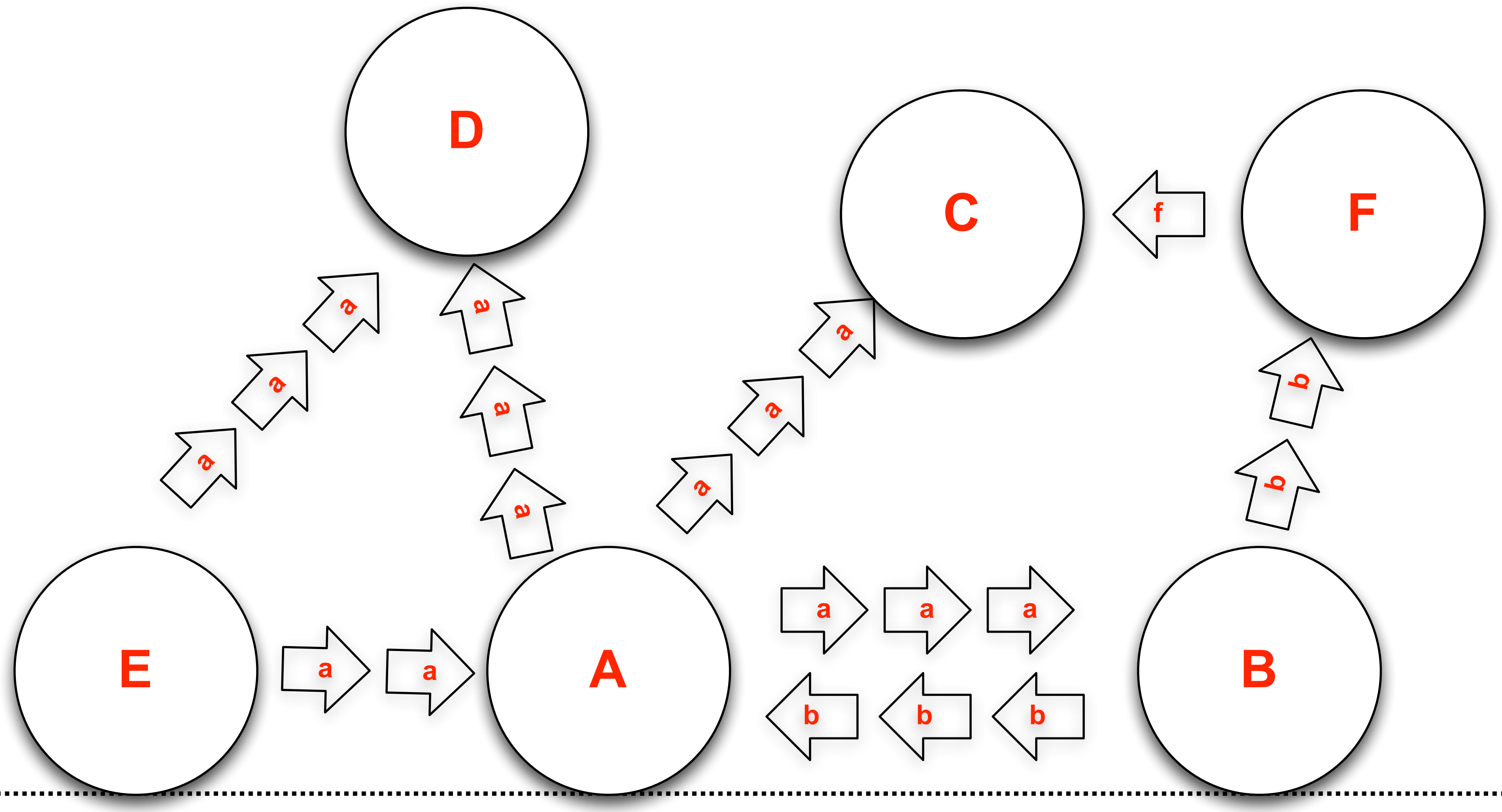


D
AC
C
conforms
E
conforms
A
partners
B

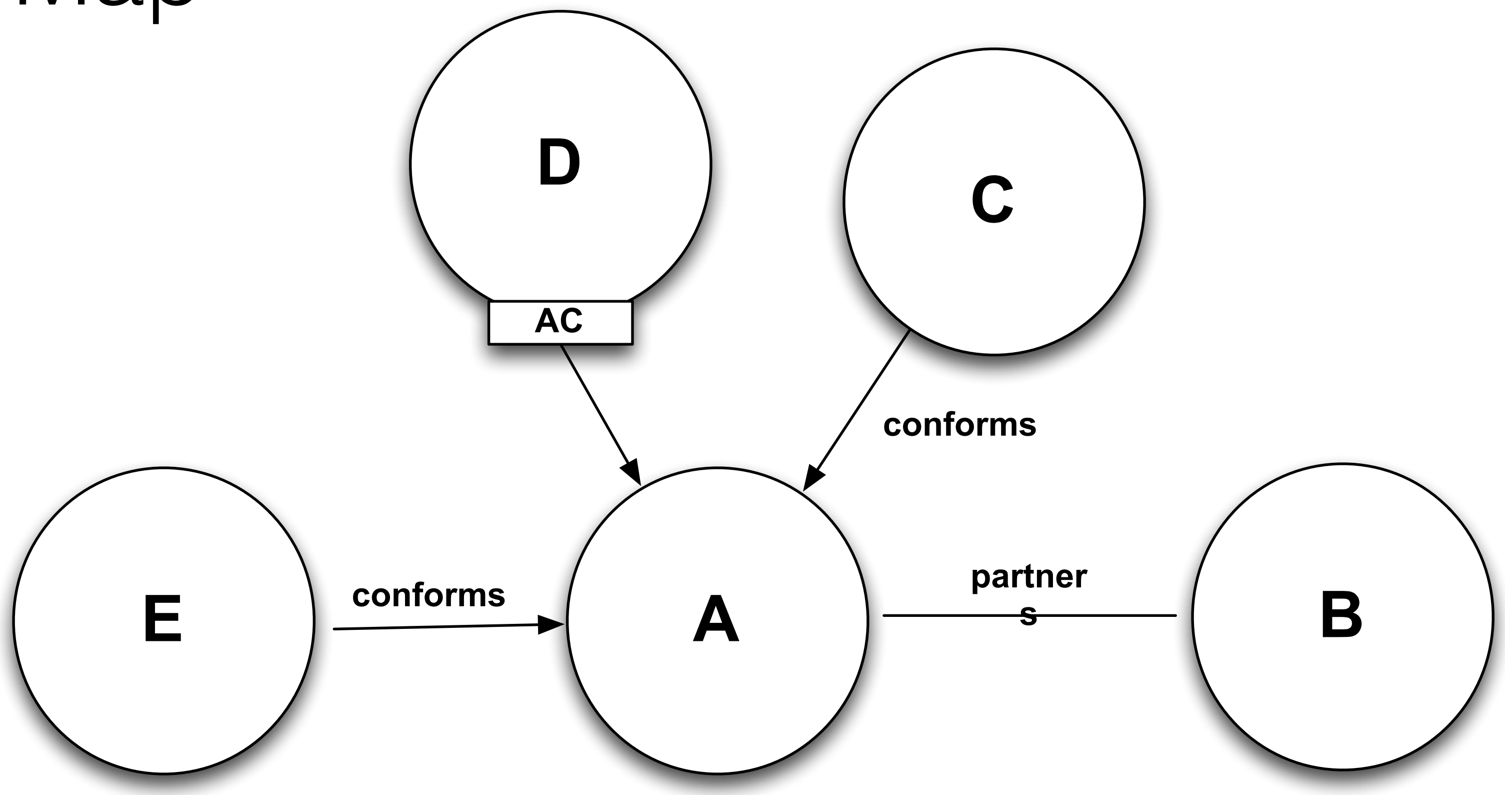


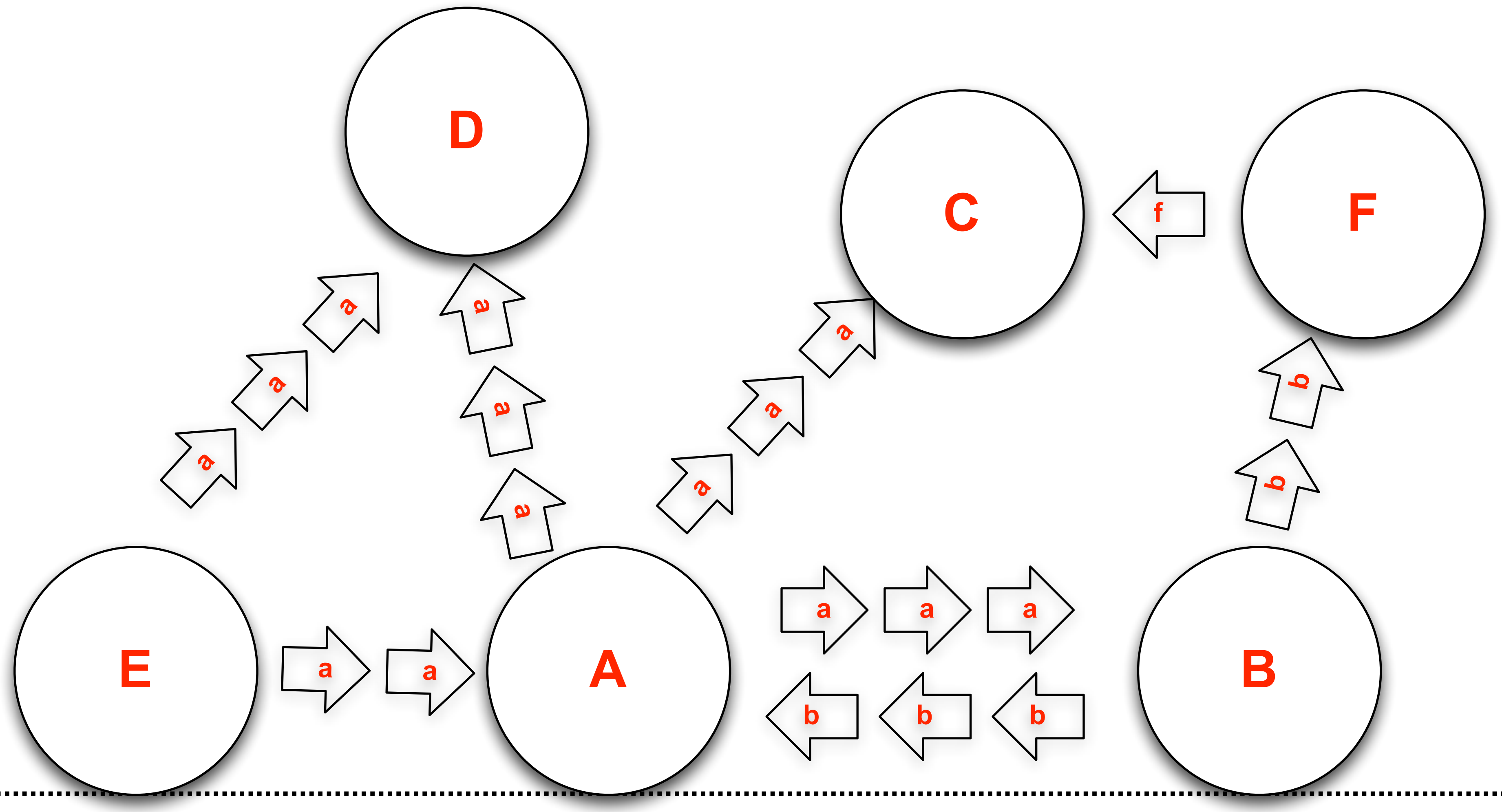
Context Map



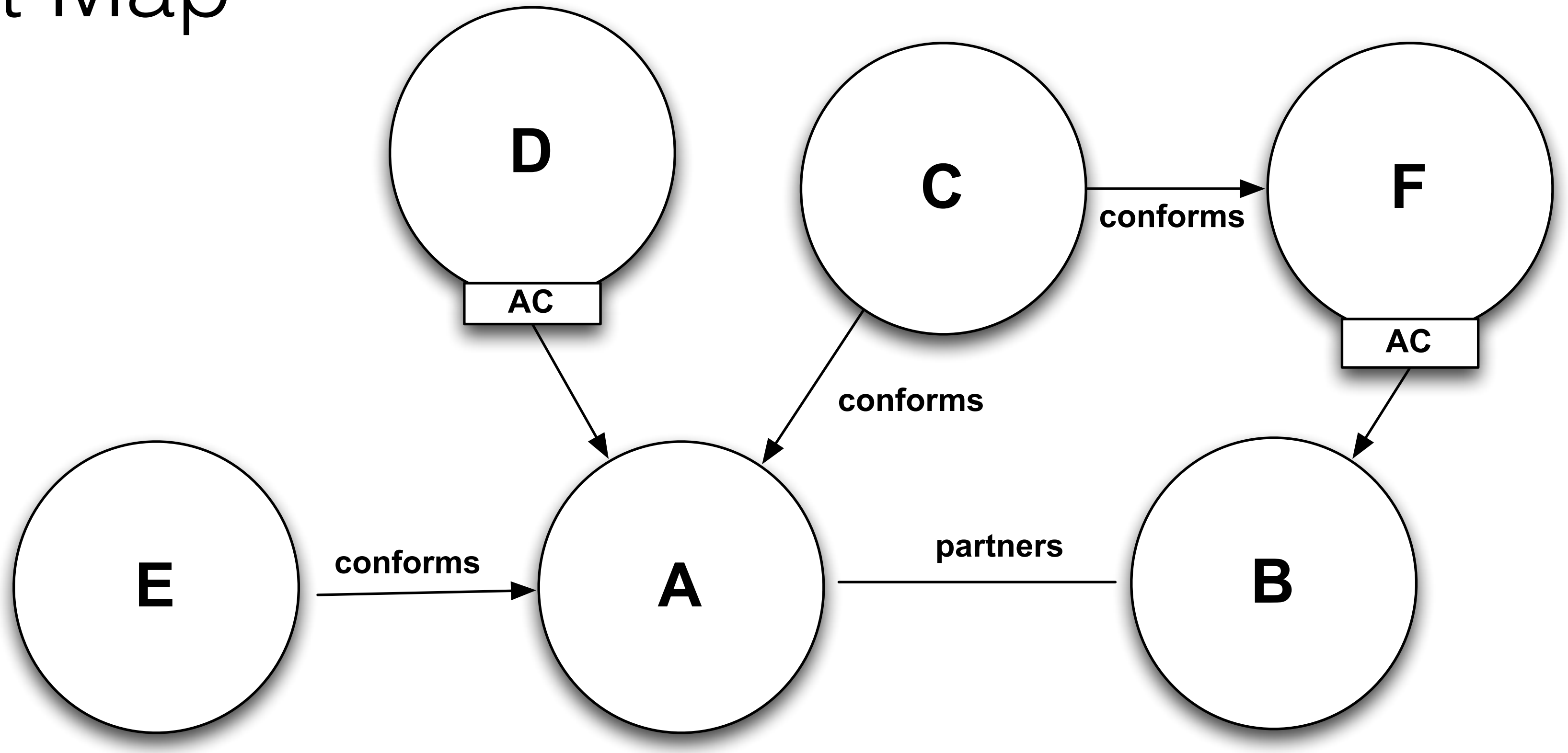

D
C
f
F
a
a
a
a
a
a
a
a
a
b
b
E
a
a
A
a
a
a
b
b
b
B

Context Map


D
AC
C
conforms
E
conforms
A
partners
B


D
C
f
F
a
a
a
a
a
a
a
a
a
b
b
E
a
a
A
a
a
a
b
b
b
B

Context Map

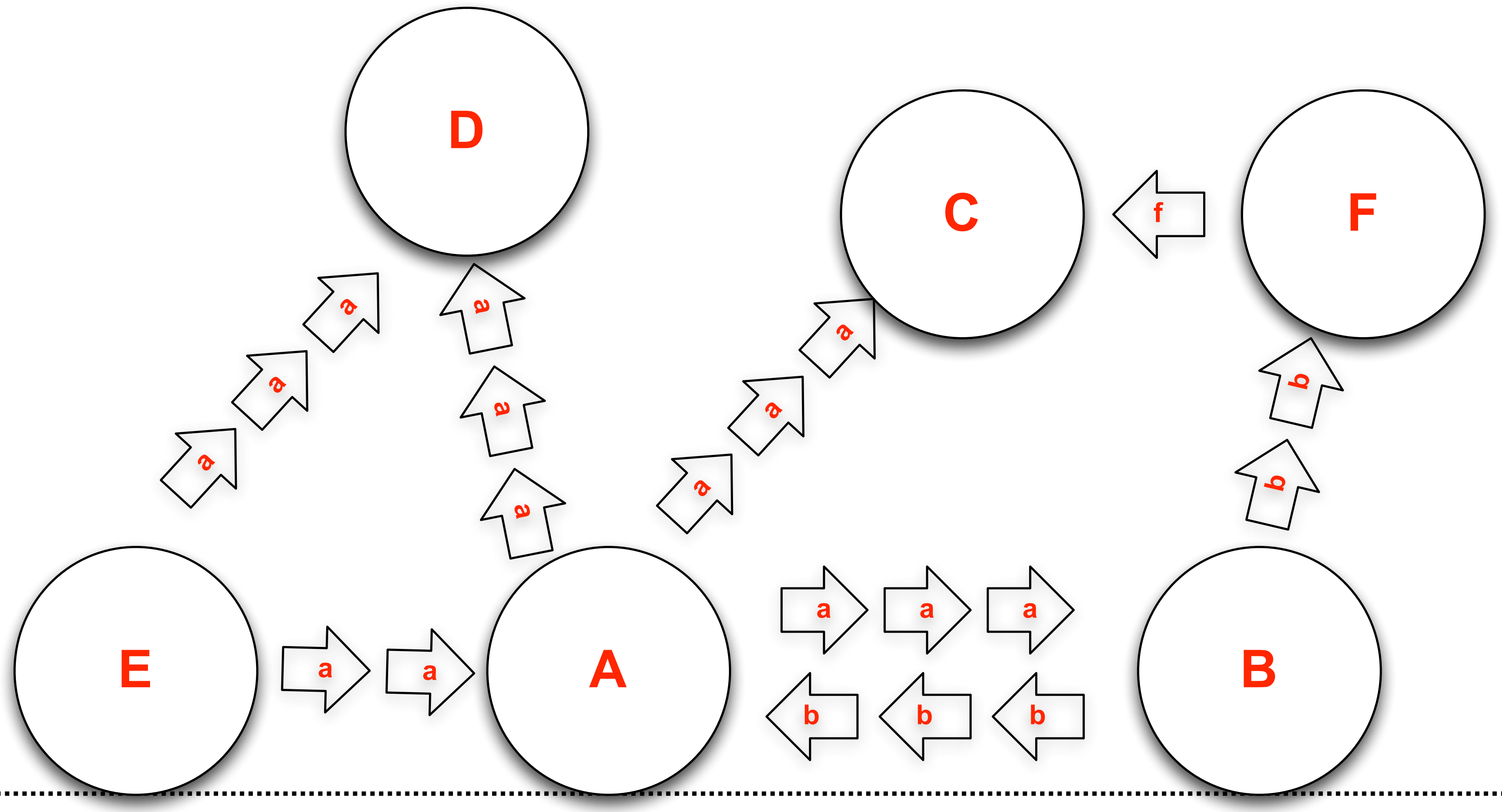

D
AC
C
conforms
F
AC
conforms
E
conforms
A
partners
B

~~enterprise model~~
~~shared database schema~~
~~unified field theory~~
~~one ring~~

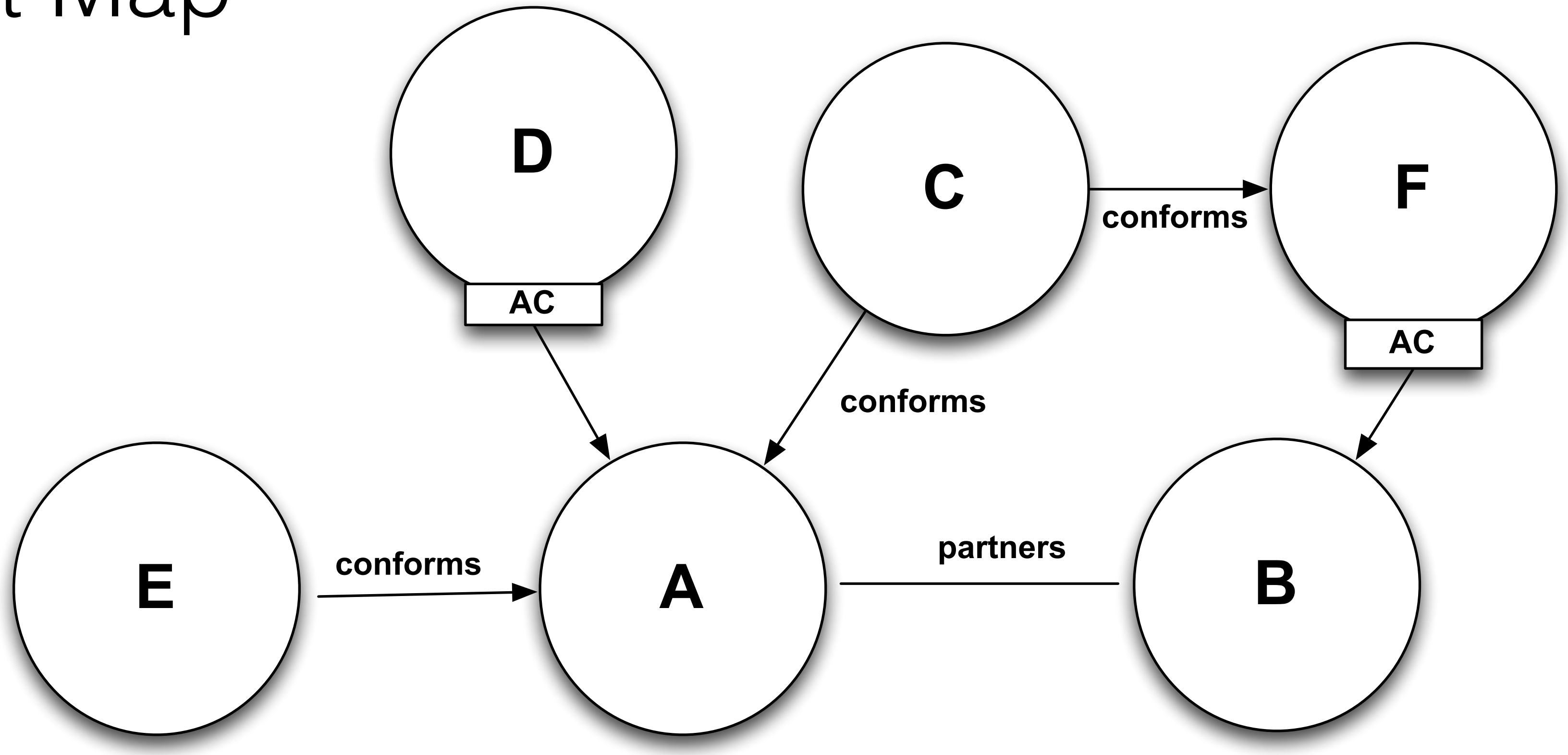
There are always multiple models.

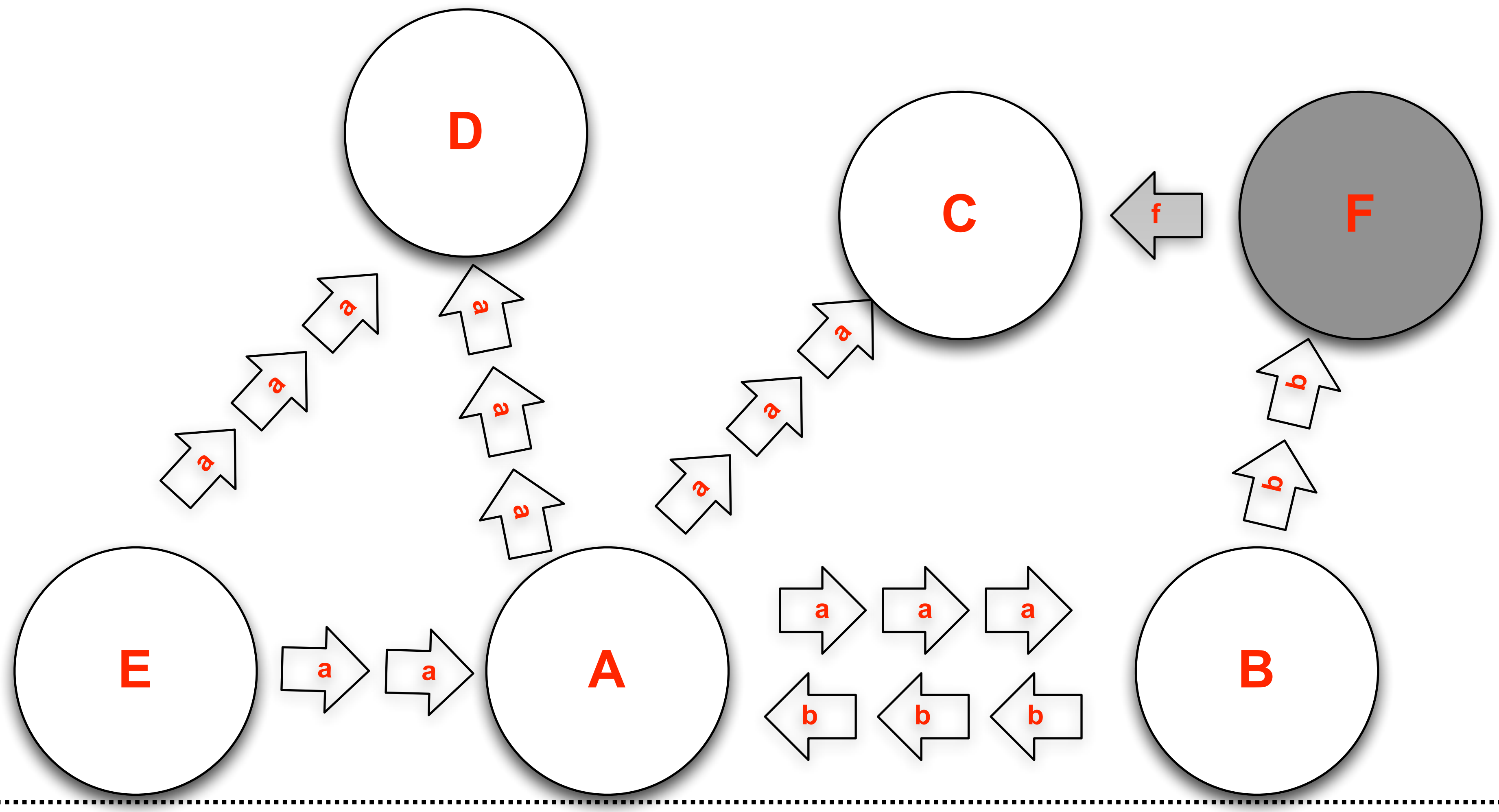
Models need to be clear, not big.

- Useful models need crisp definitions.
- Definitions require clear context.
- Useful models need simple assertions.
- *Assertions require boundaries*.

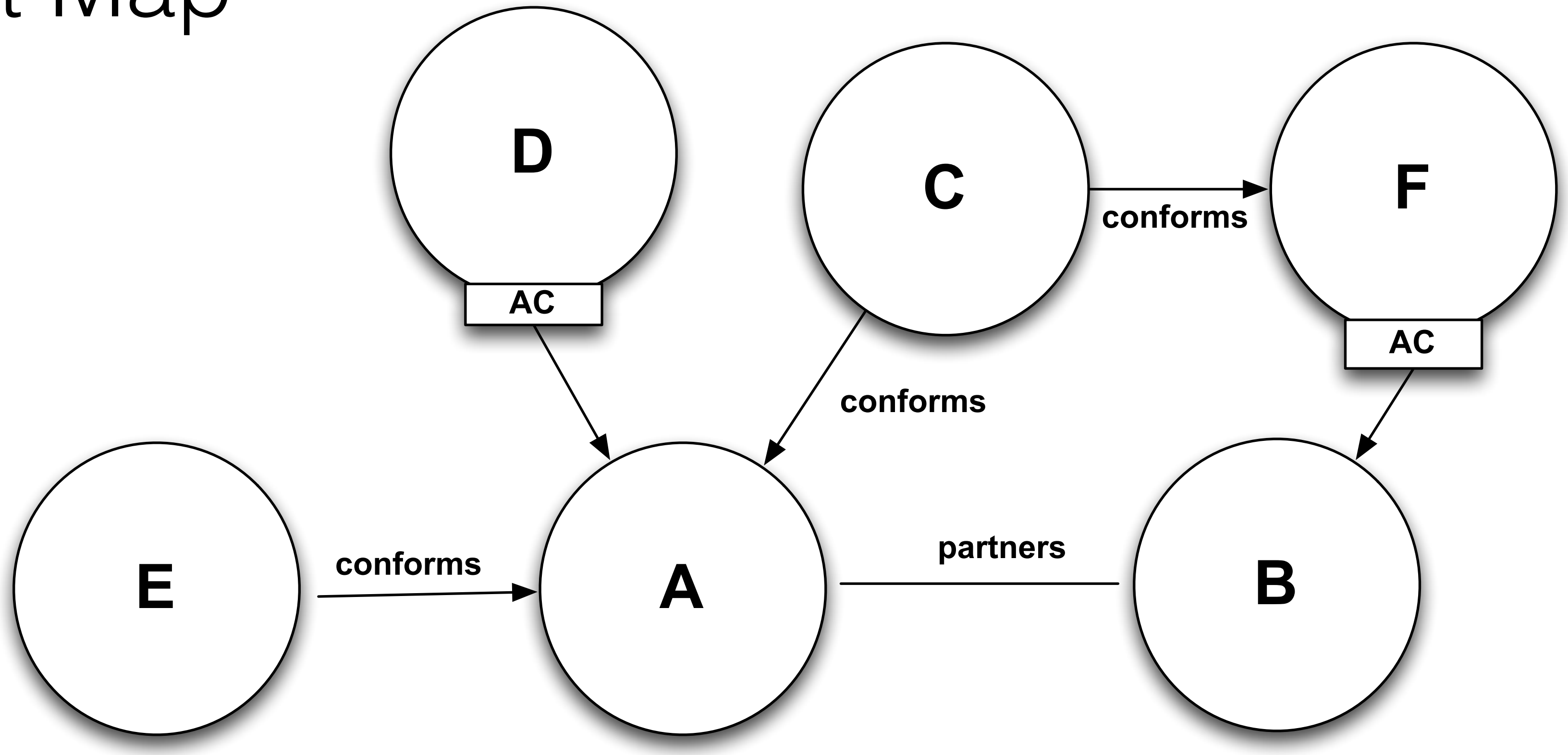

D
C
F
E
A
B
a
b
f

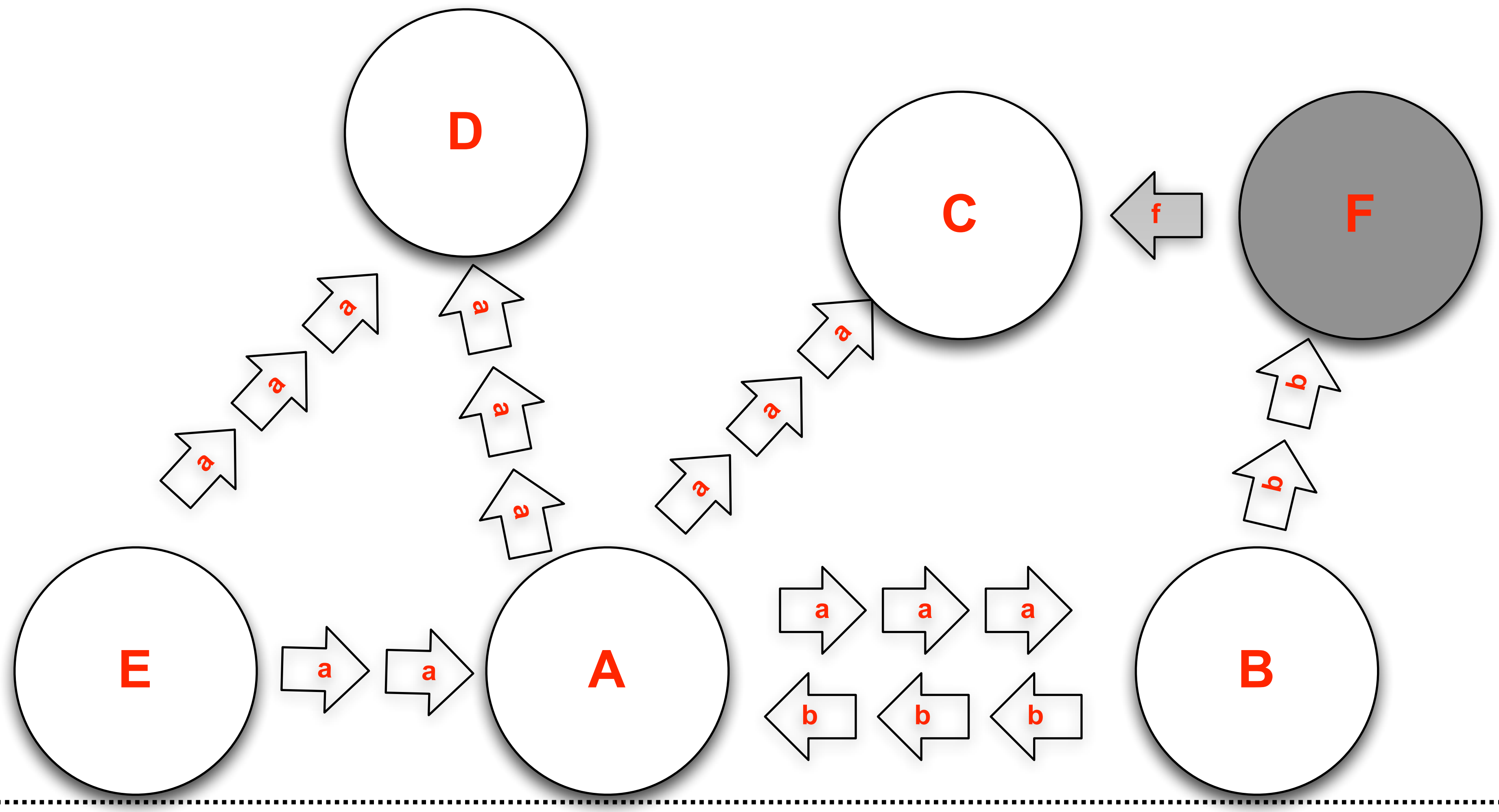
Context Map


D
AC
C
conforms
F
AC
conforms
E
conforms
A
partners
B

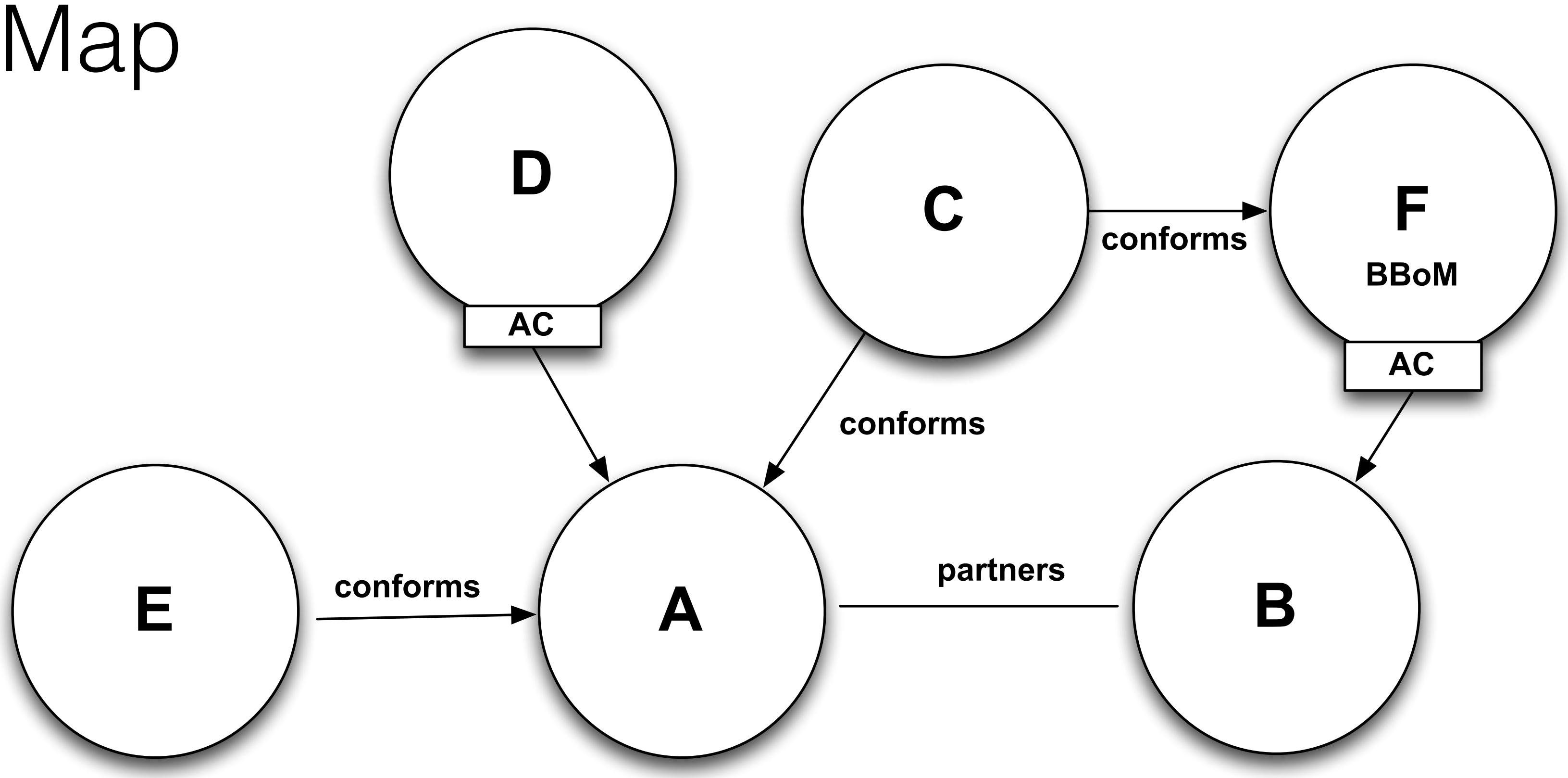

D
C
F
E
A
B
a
b
f

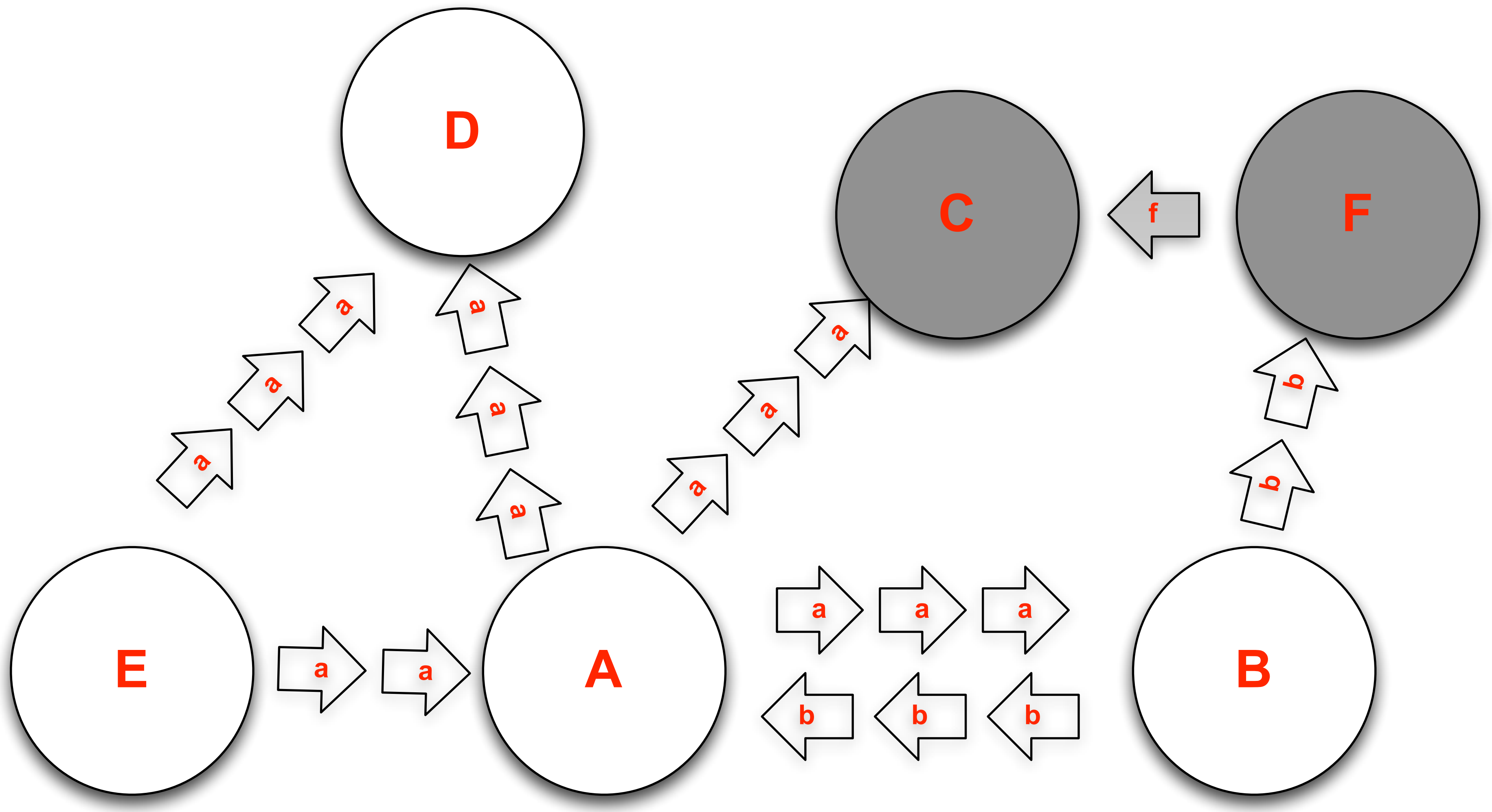
Context Map


D
AC
C
conforms
F
AC
conforms
E
conforms
A
partners
B

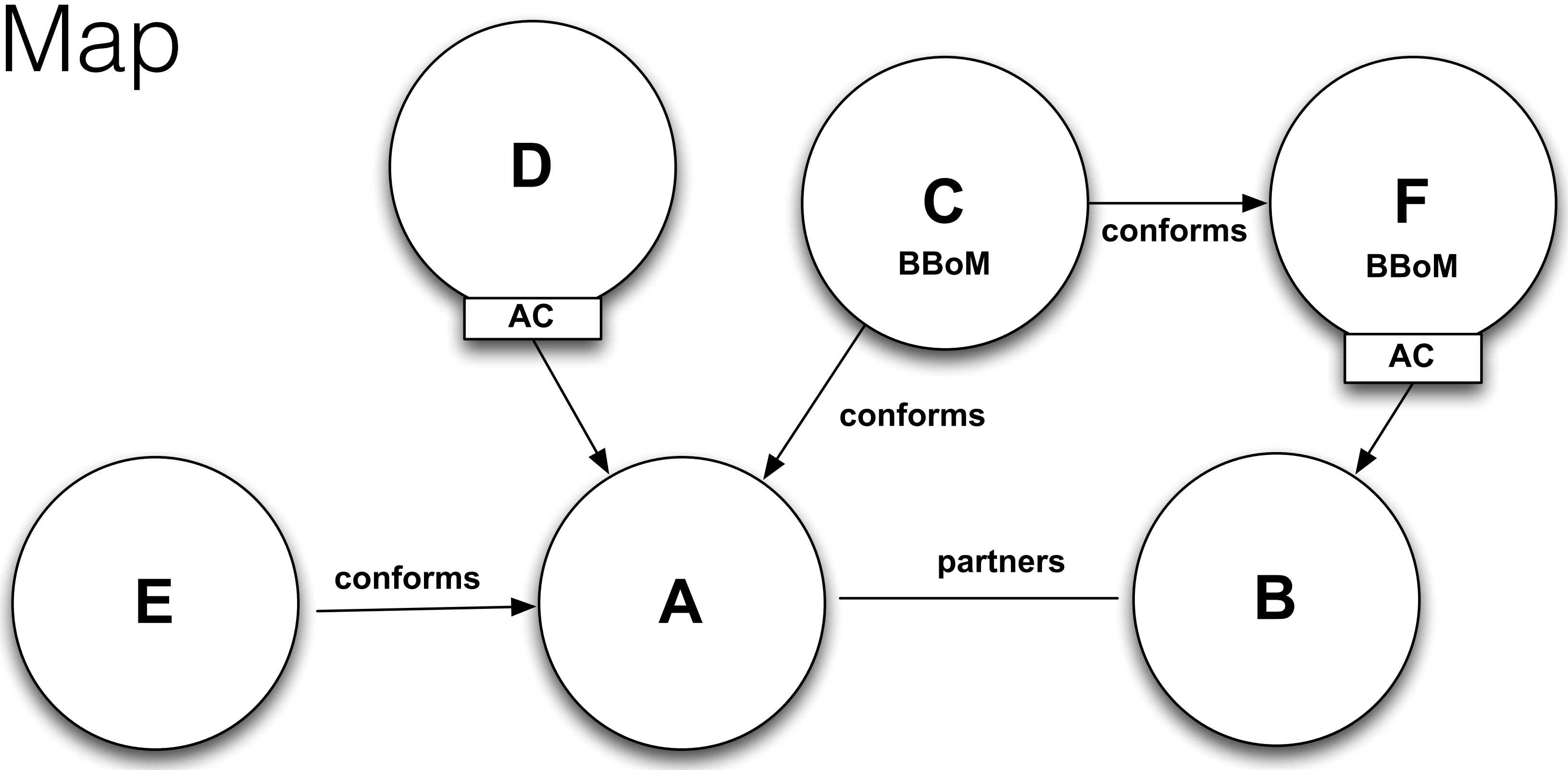

D
C
F
f
a
a
a
a
a
a
a
a
a
b
b
E
a
a
A
a
a
a
b
b
b
B

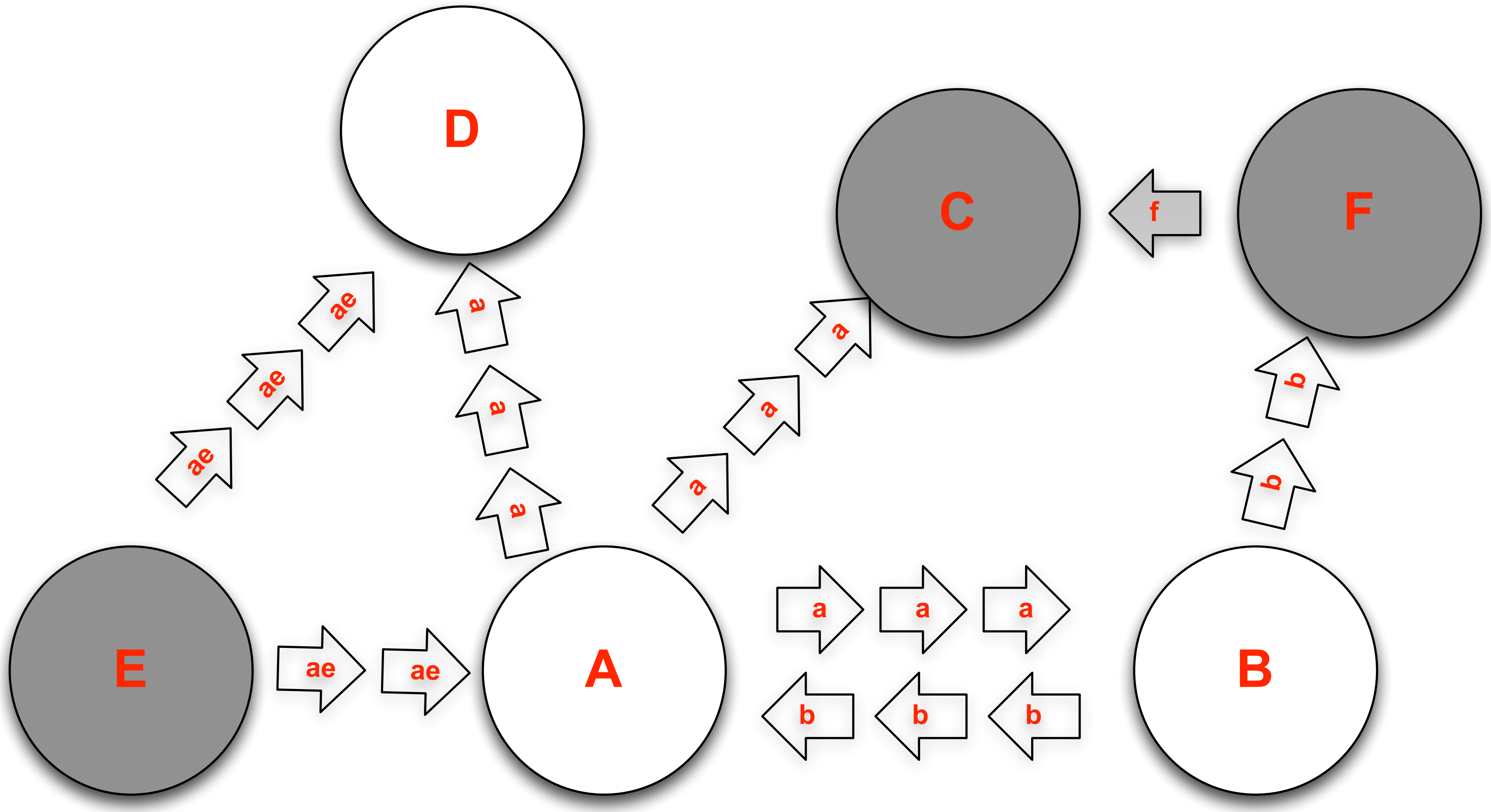
Context Map


D
AC
C
conforms
F
BBoM
AC
conforms
E
conforms
A
partners
B


D
C
F
f
a
a
a
a
a
a
a
a
a
b
b
E
a
a
A
a
a
a
b
b
b
B

Context Map

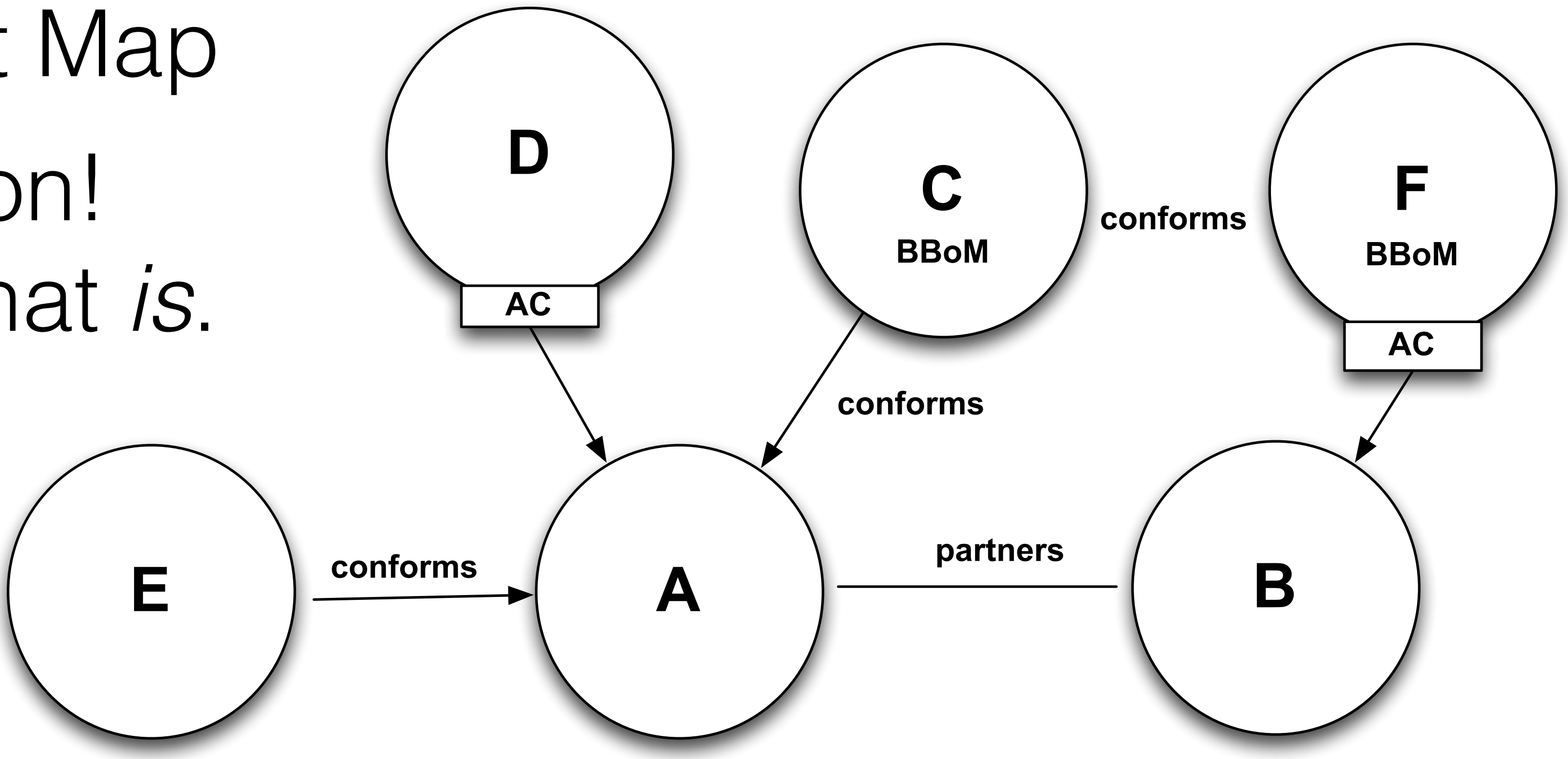

D
AC
C
BBoM
conforms
F
BBoM
AC
conforms
E
conforms
A
partners
B

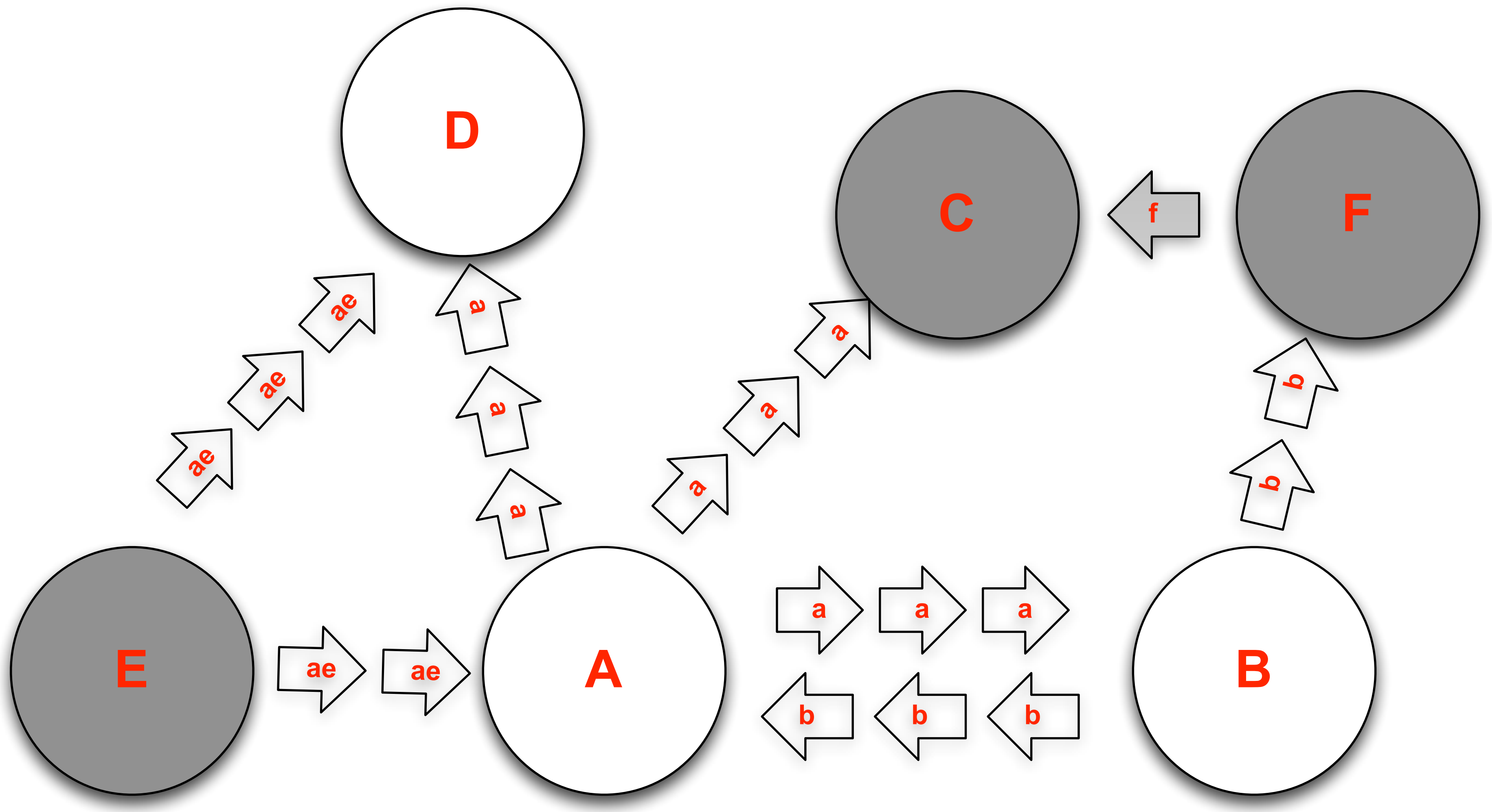


Context Map

Fiction!

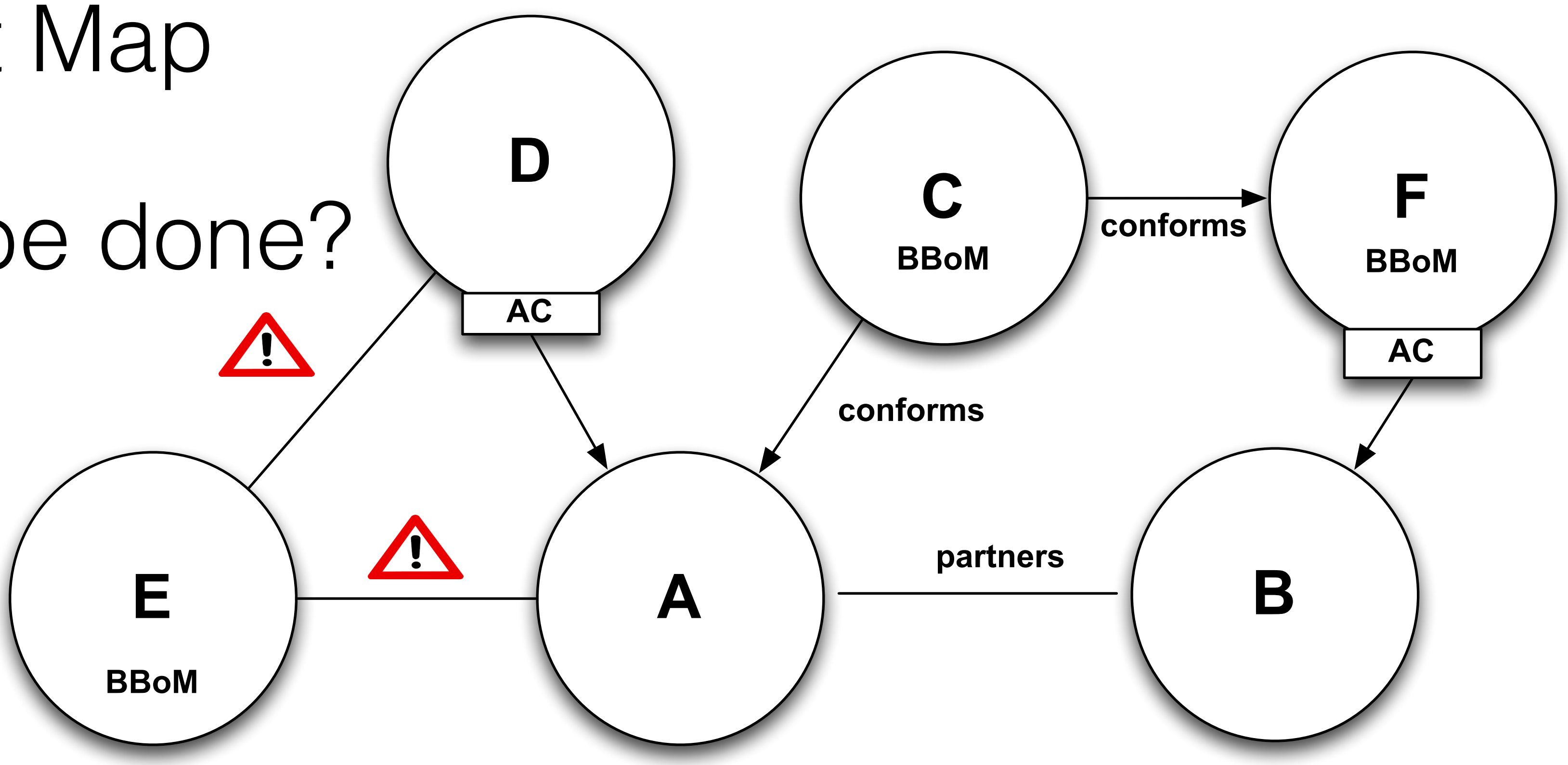
Map what *is*.



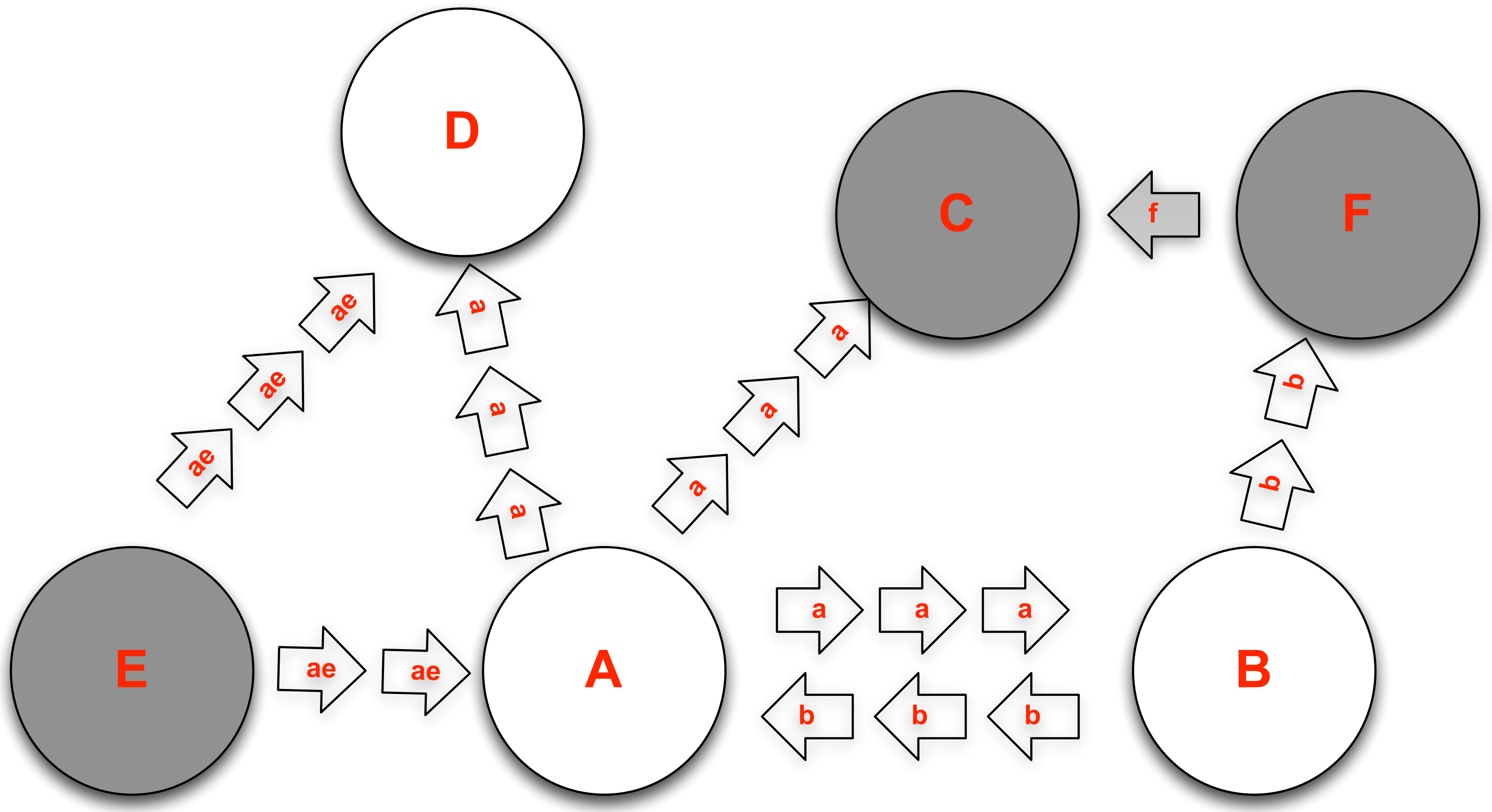


Context Map

What can be done?

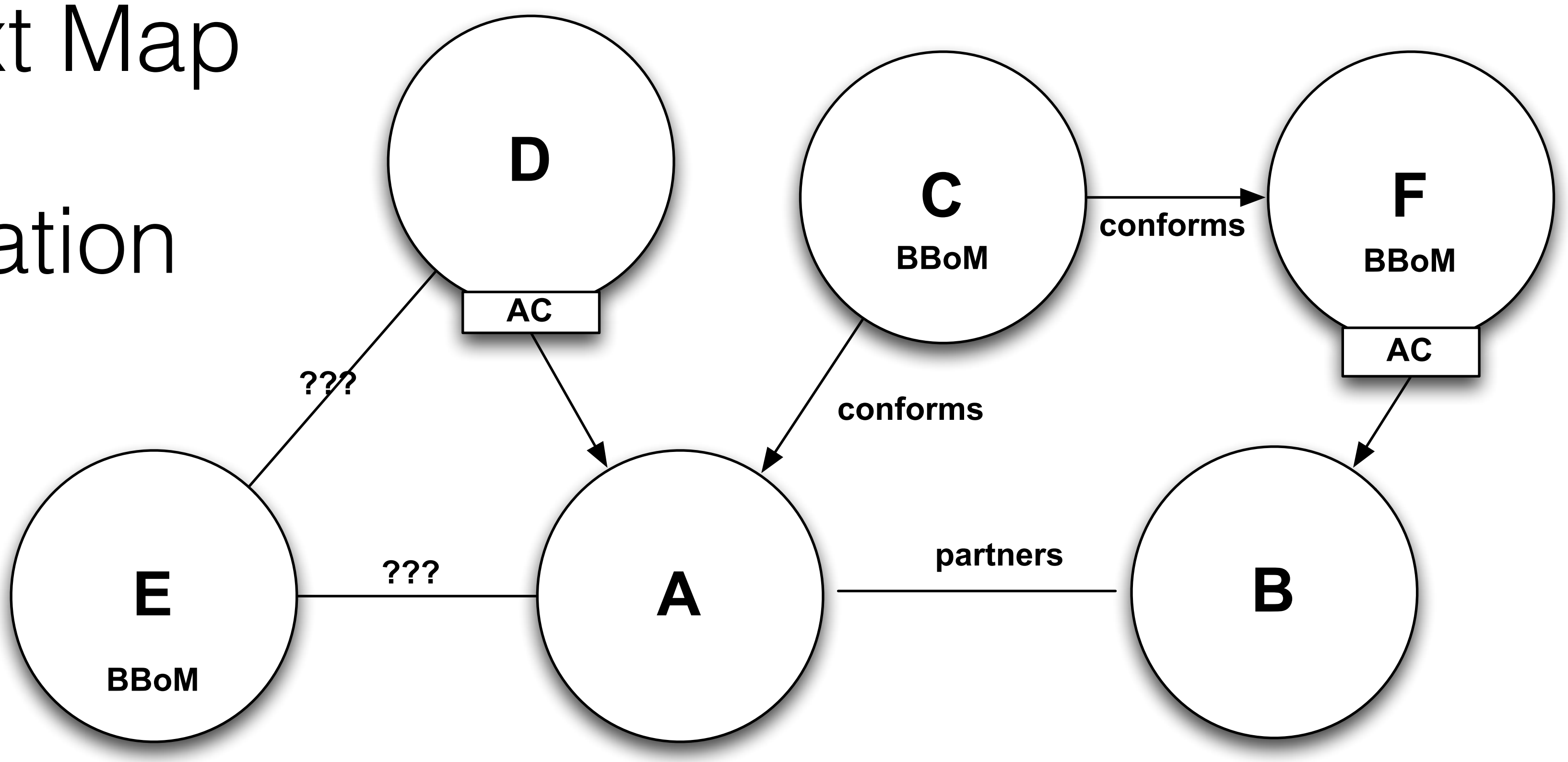


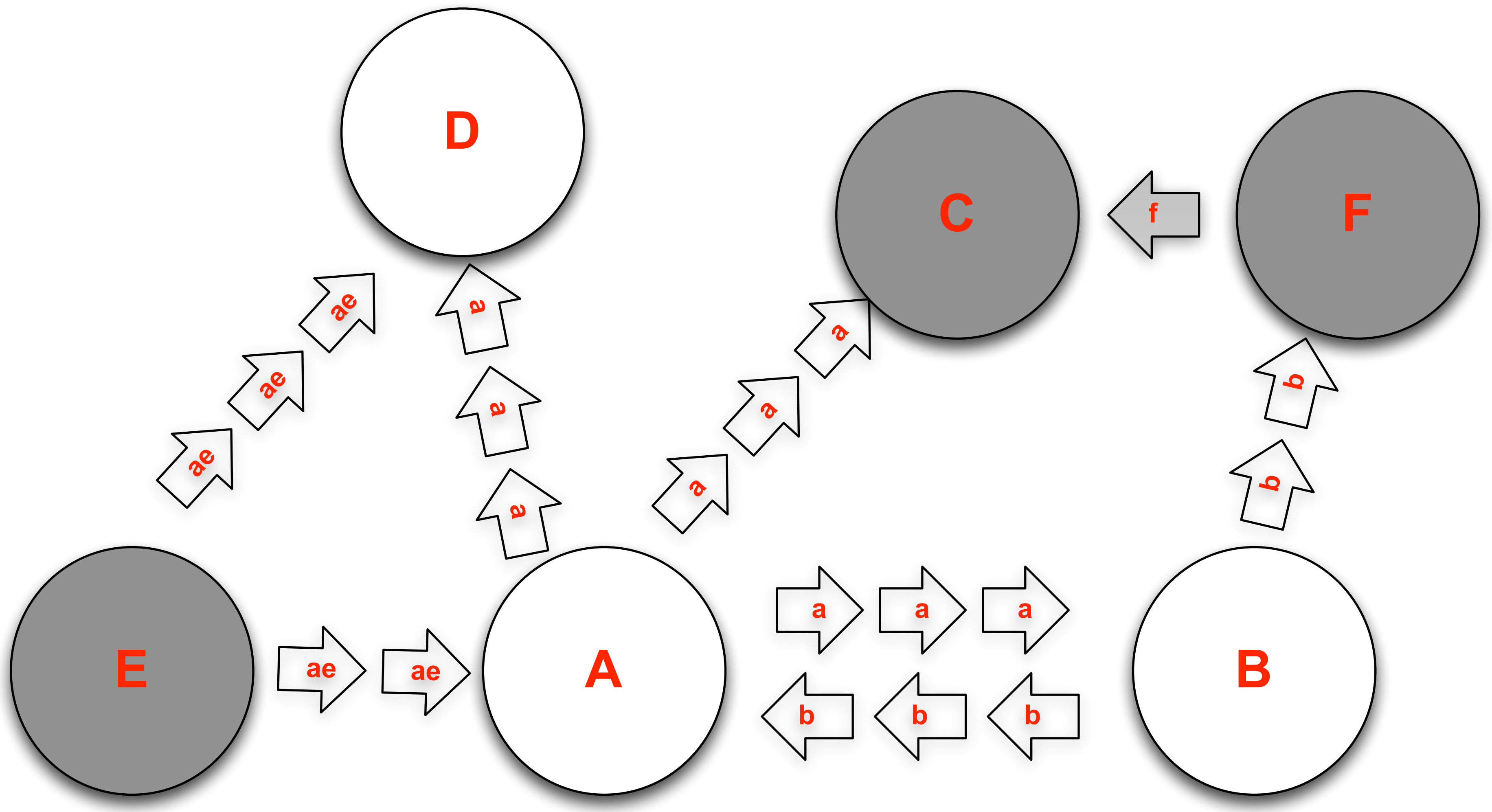
Not all of a large system will be well designed.



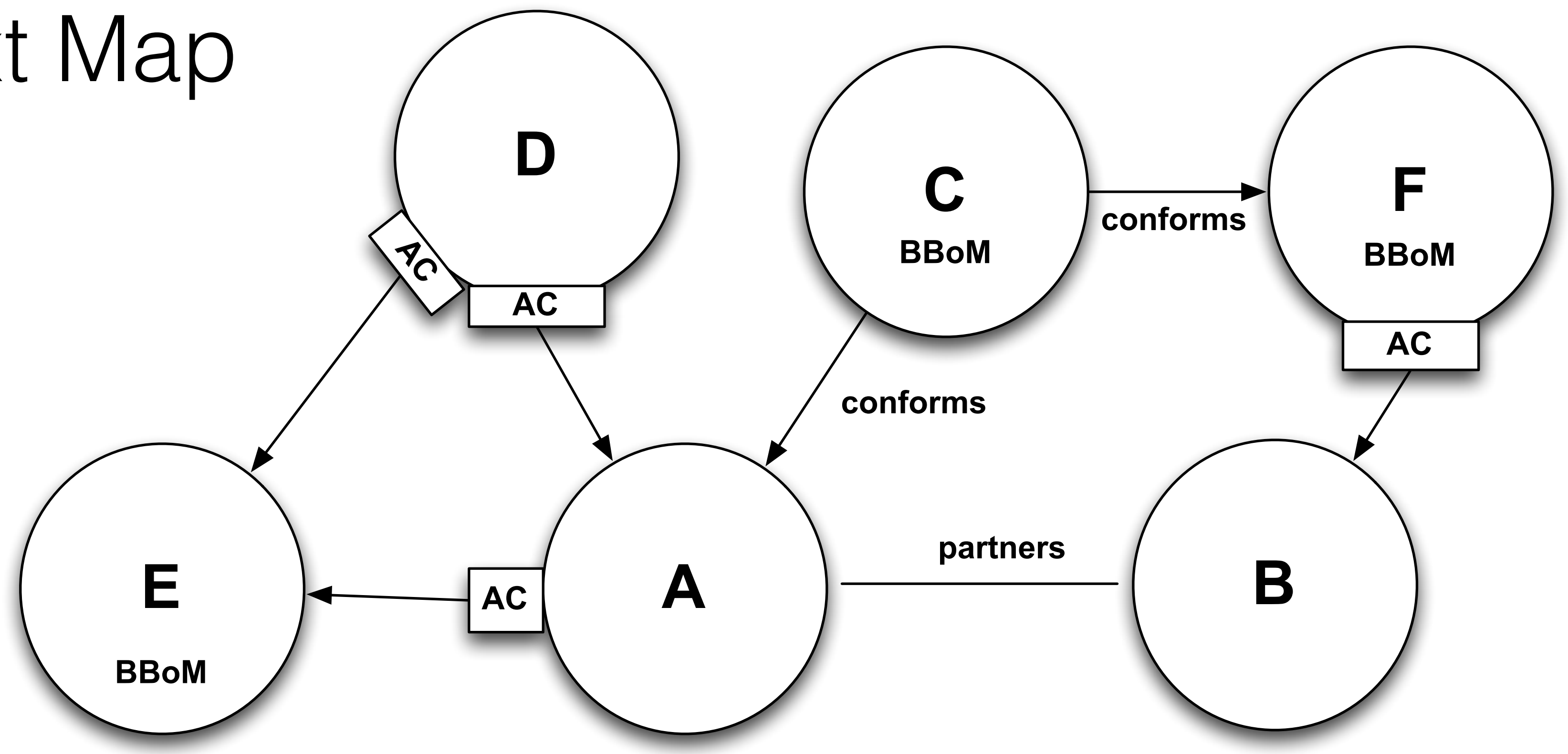
Context Map

Mitigation




D
C
F
E
A
B
ae
a
b
f

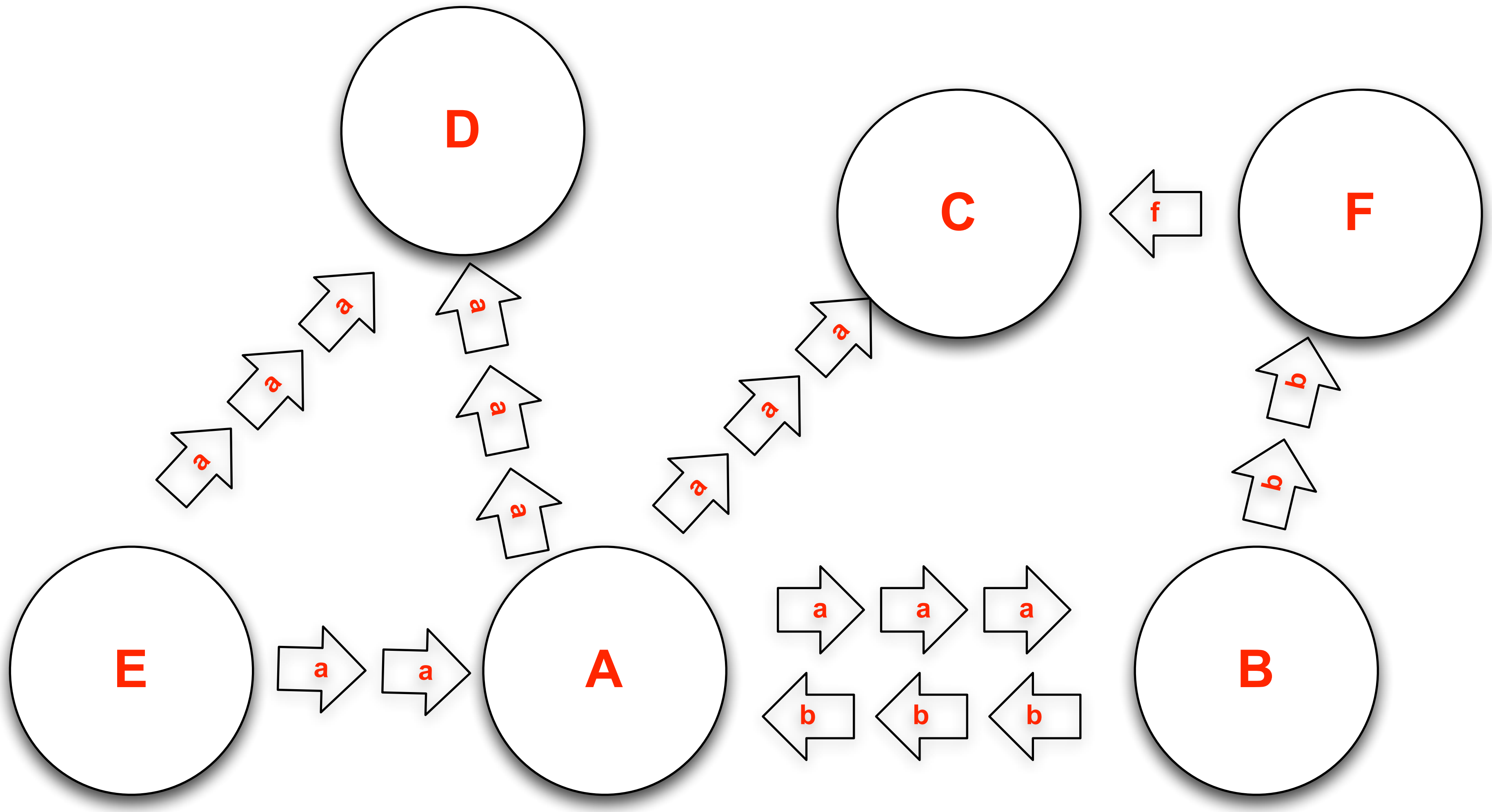
Context Map


D
AC
AC
C
BBoM
conforms
F
BBoM
AC
conforms
E
BBoM
AC
A
partners
B

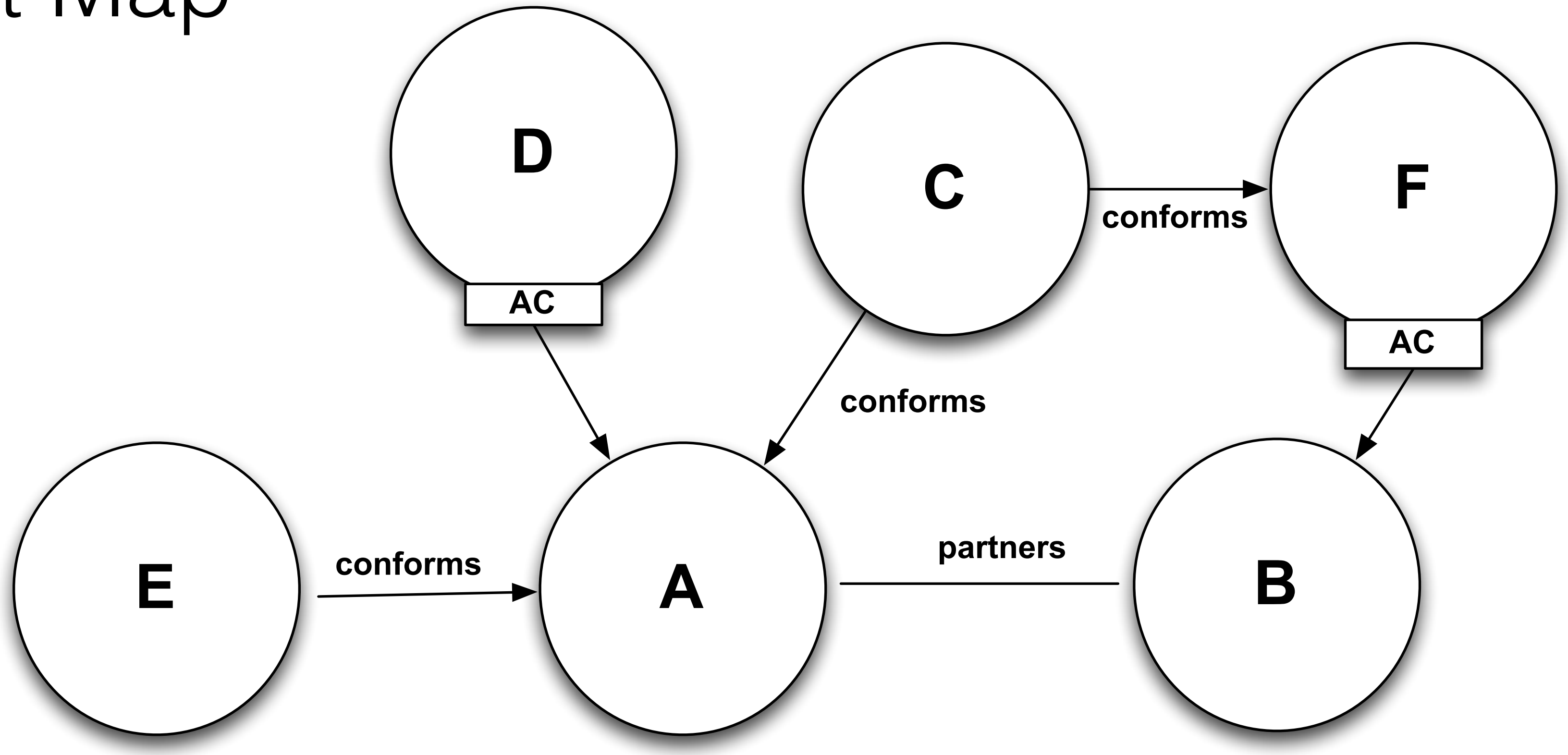
Microservice as Context Boundary

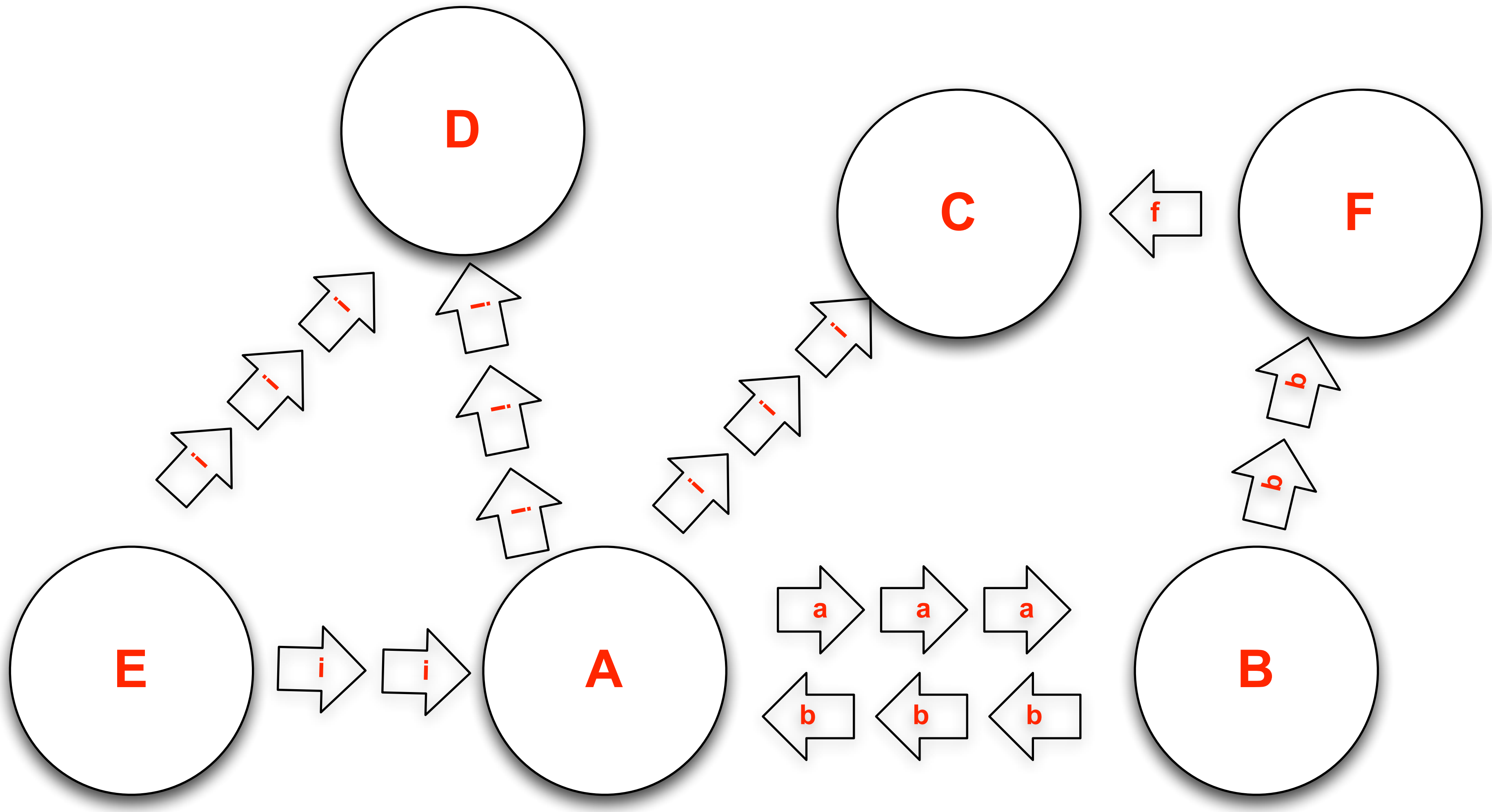
- Allow high-concept modeling in a messy world.
- Allow specialized models for distinct problems.
- Mitigate consequences of design mistakes.
- Acknowledge the rough and tumble of enterprises.
- But…
- Very interesting stuff is not inside the services!

Interchange context

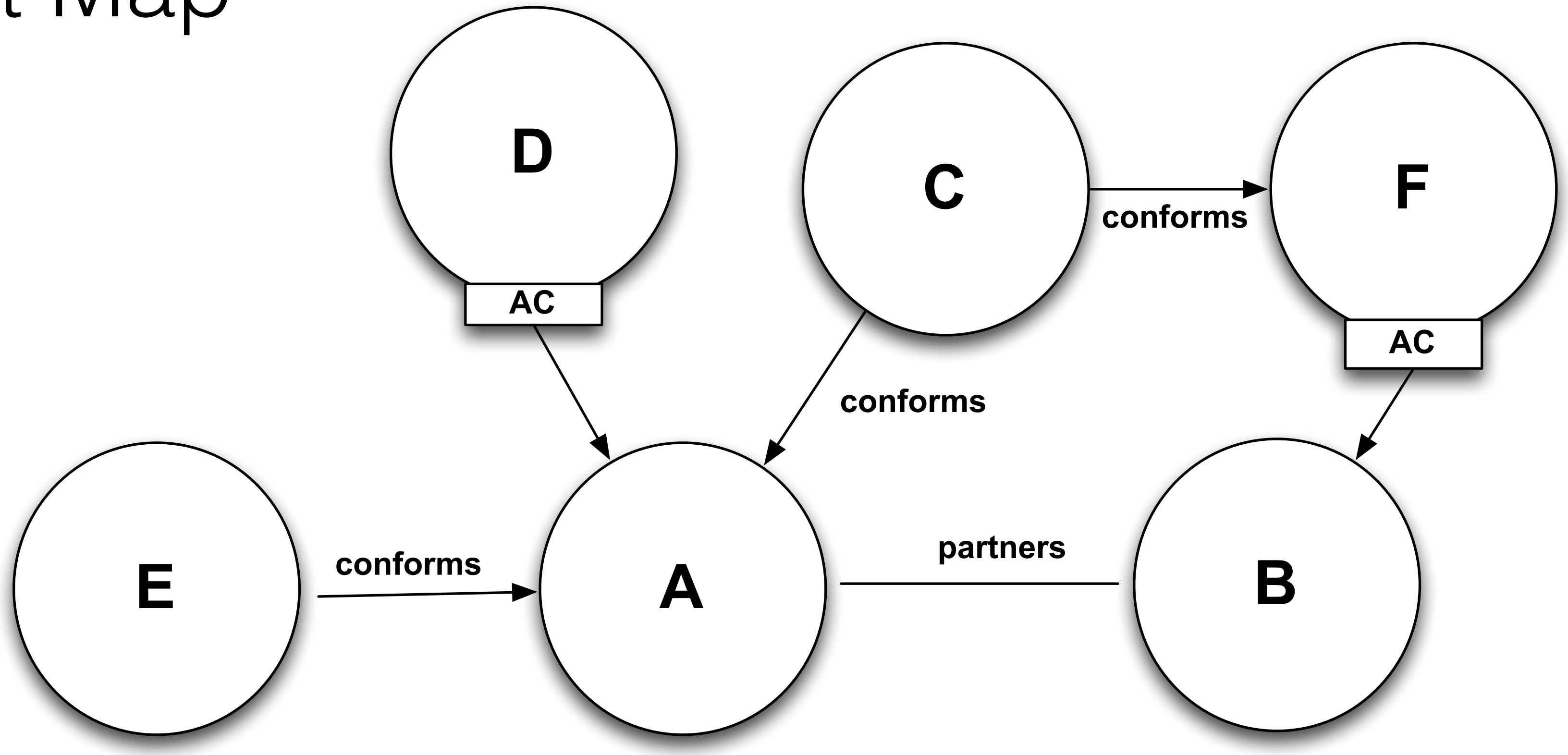

D
C
F
E
A
B
a
b
f

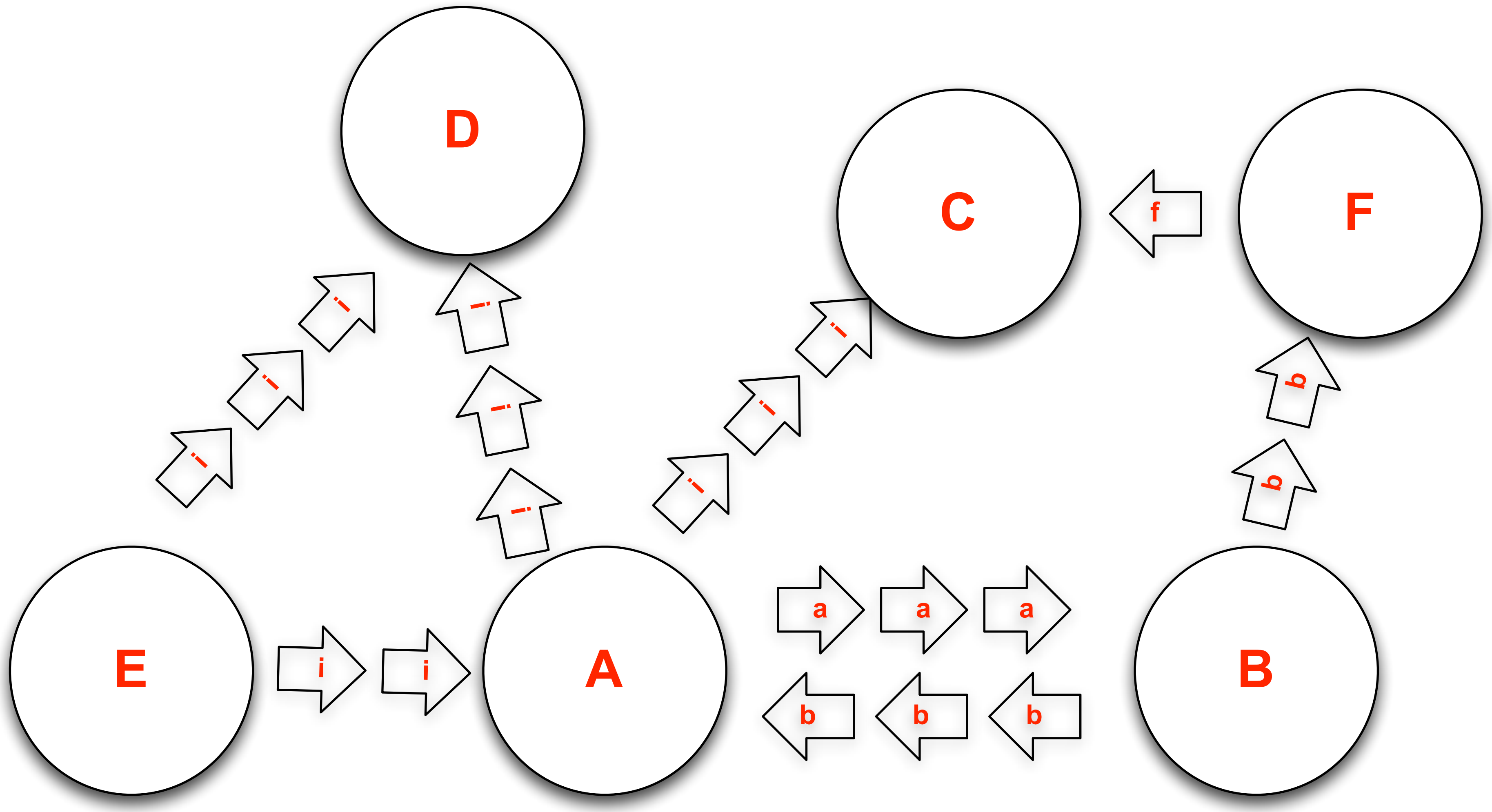
Context Map


D
AC
C
conforms
F
AC
conforms
E
conforms
A
partners
B

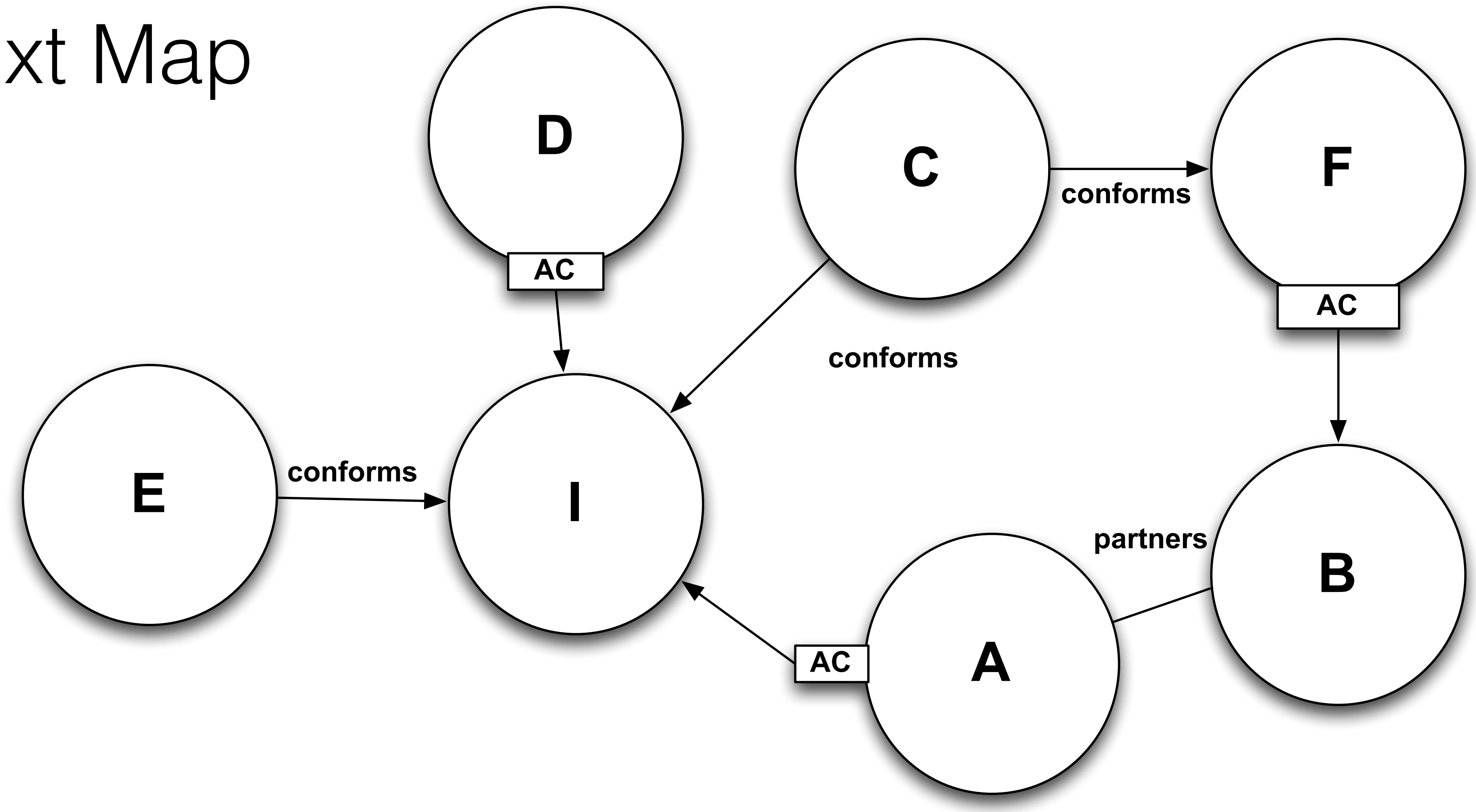

D
C
F
E
A
B
i
a
b
f

Context Map


D
AC
C
conforms
F
AC
conforms
E
conforms
A
partners
B


D
C
F
E
A
B
i
a
b
f
g

Context Map


D
AC
C
conforms
F
AC
conforms
E
conforms
I
partners
B
AC
A

`

- A relatively generic data model for sharing.
- or…
- A place to model *protocols of interaction*.
- Modeling and design of higher-level solutions.
- A domain language tuned to these purposes.

Interchange Context

- Expressed in terms of service interfaces/messages.
- Distinct from the objects/functions of the internals of a service.
- Prevents distortion/freezing of early-dominant contexts.
- Gives big-picture understanding when we have many services.
- Usually more than one! (Avoid enterprise model.)

Why not logical boundaries?

- Smart people I respect point out that most of what I want is the logical partitioning of the system.
- We’ve had decades to get that to work.
- Some techniques are too subtle to survive the rough and tumble.

Wrap up

- Subtle design (such as DDD) requires concrete boundaries. Microservices have them.
- Proliferation of services recreate some of the old problems.
- Context Maps help visualize and communicate about those problems.
- *Modest* use of interchange contexts can help produce coherent sets of microservices.

Not all of a large system
will be well designed.

DDD & Microservices

At last, some boundaries!

Eric Evans
@ericevans0
domainlanguage.com