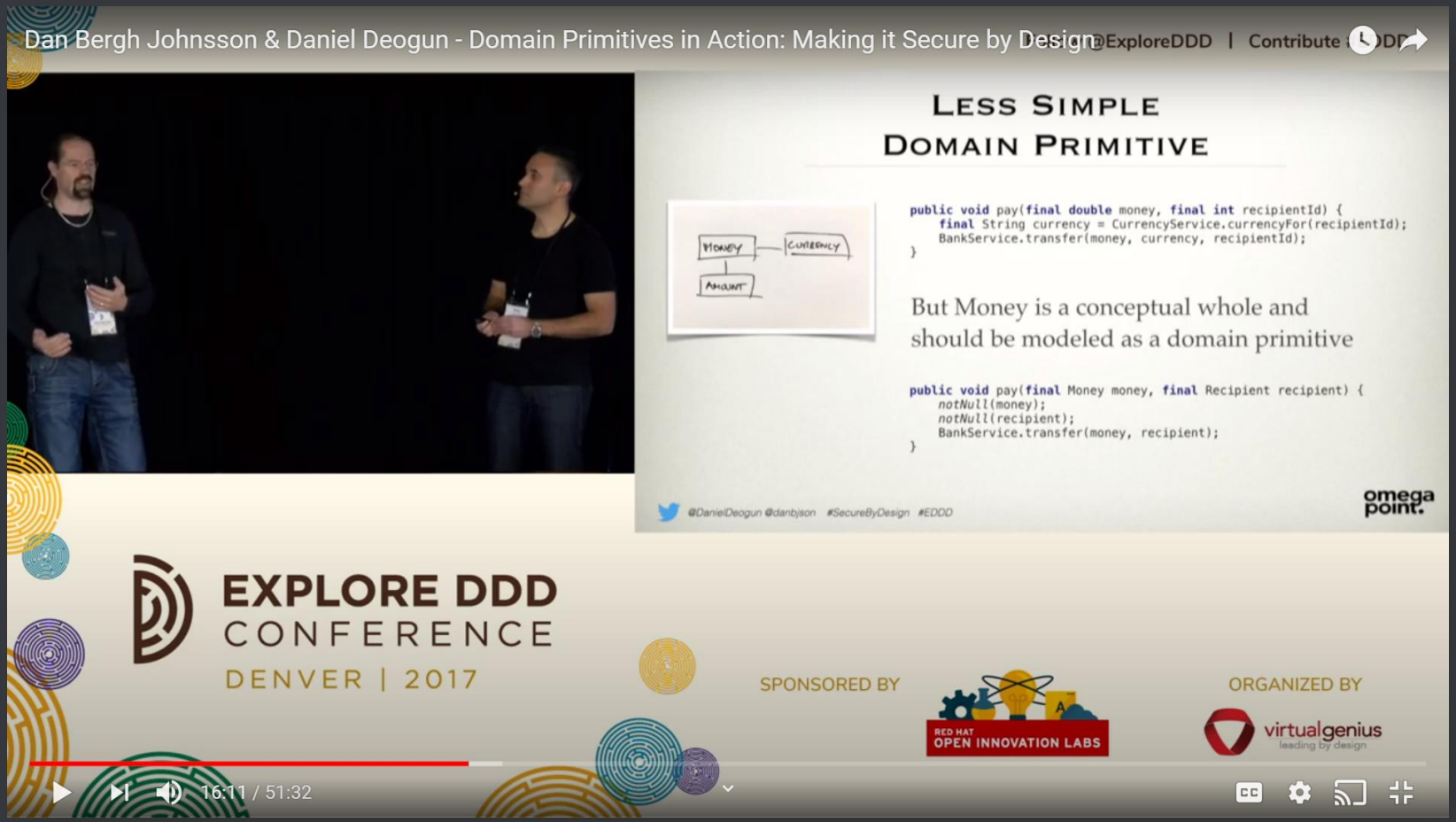

Dan Bergh Johnsson & Daniel Deogun - Domain Primitives in Action: Making it Secure by Design
@ExploreDDD | Contribute
LESS SIMPLE
DOMAIN PRIMITIVE
Money
Currency
Amount
public void pay(final double money, final int recipientId) {
final String currency = CurrencyService.currencyFor(recipientId);
BankService.transfer(money, currency, recipientId);
}
But Money is a conceptual whole and should be modeled as a domain primitive
public void pay(final Money money, final Recipient recipient) {
notNull(money);
notNull(recipient);
BankService.transfer(money, recipient);
}
@DanielDeogun @danbjson #SecureByDesign #EDDD
omega point.
EXPLORE DDD
CONFERENCE
DENVER | 2017
SPONSORED BY
RED HAT
OPEN INNOVATION LABS
ORGANIZED BY
virtualgenius
leading by design
16:11 / 51:32

4. Juni 1996

"However, problems began to occur when the software attempted to stuff this 64-bit variable (...) into a 16-bit integer. (...) For the first few seconds of flight, the rocket's acceleration was low, so the conversion between these two values was successful. However, as the rocket's velocity increased, the 64-bit variable exceeded 65k, and became too large to fit in a 16-bit variable."

23. September 1999

"Quality Assurance had not found the use of an imperial unit in external software, despite the fact that NASA's coding standards at the time mandated use of metric units."

- A value object precise enough in its definition that it, by its mere existence, manifests its validity is called a **domain primitive**.

R. Clark (2019)

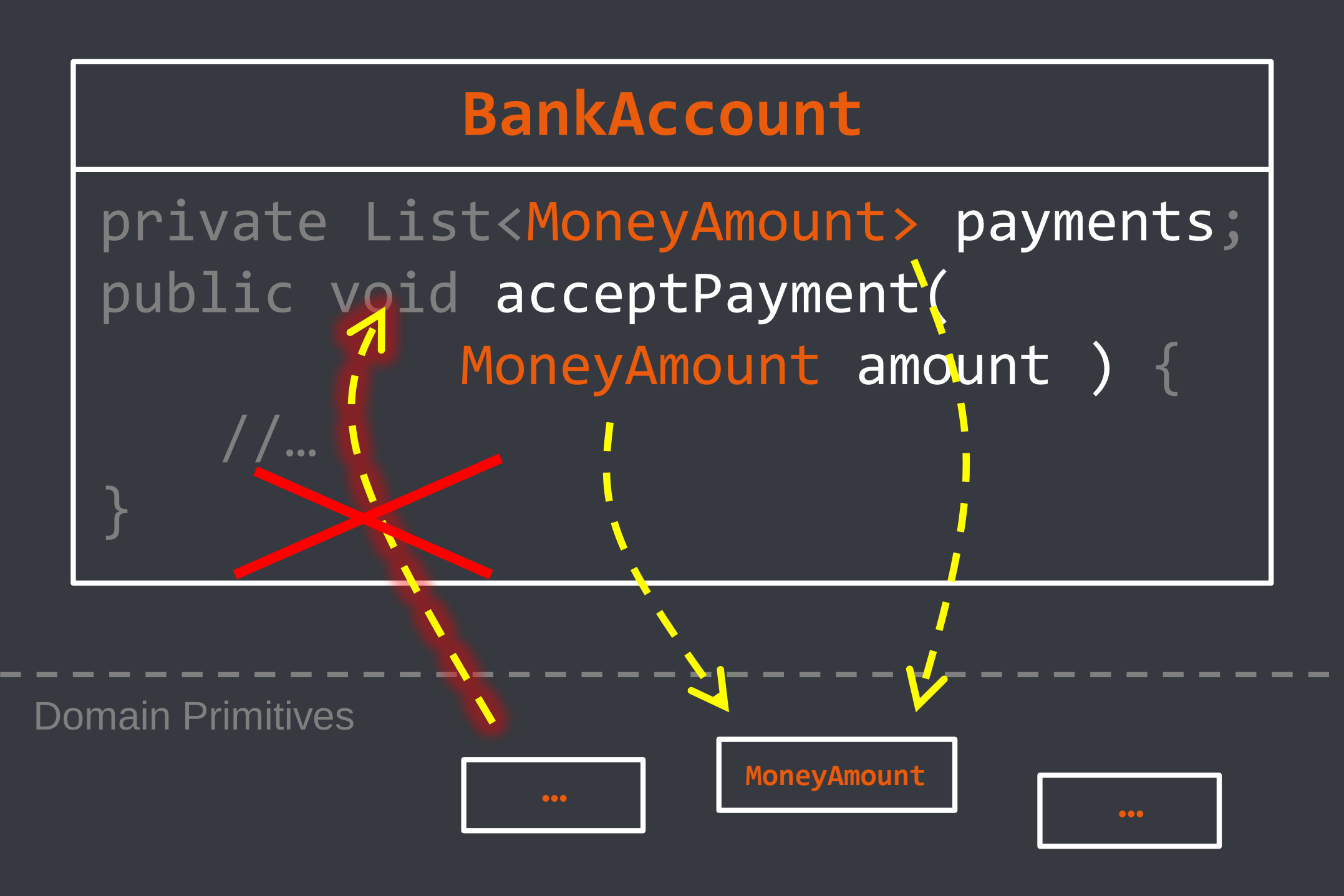
  - […] we require invariants to exist and they must be enforced at the point of creation.

D. B. Johnsson, D. Deogun, D. Sawano (2018)

1. DPs can only exist if they're valid.
2. Invariants are checked at creation.
3. Should always be used instead of language primitives or generic types.
4. Their meaning's defined within the boundaries of the current domain, even if the same term exists outside of the current domain.

D. B. Johnsson, D. Deogun, D. Sawano (2018)

1. Value Object
   - keine Setter
   - Änderungen geben neues Objekt zurück (immutable)
2. Erzeugung über Factory-Methode
   - `protected` oder `private` Constructor
   - `valueOf(…)`, `fromXXX(…)`
3. Exceptions für Fehler bei Erzeugung
4. Keine Rückreferenzen auf Domain-Entities


BankAccount
private List<MoneyAmount> payments;
public void acceptPayment(
MoneyAmount amount) {
//…
}
Domain Primitives
…
MoneyAmount
…

www.archi-lab.io