
Authenticity by Typing for Security Protocols

Andrew D. Gordon
Microsoft Research

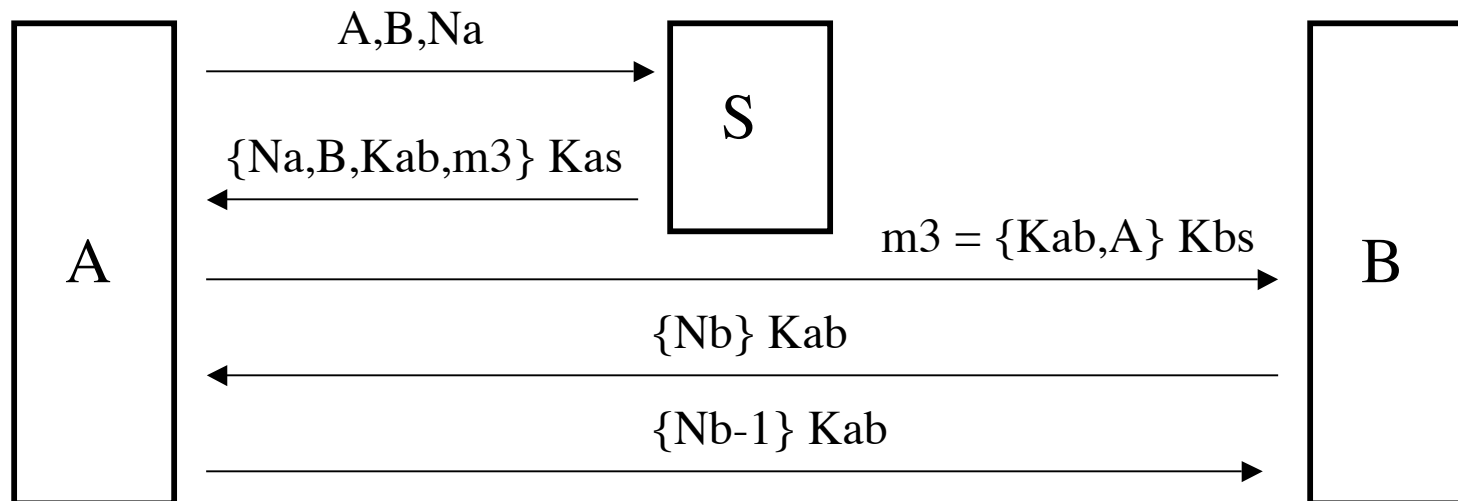
Alan Jeffrey
DePaul University

*Presented for CMSC838z by
Saurabh Srivastava*

The way we (used to?) make protocols:

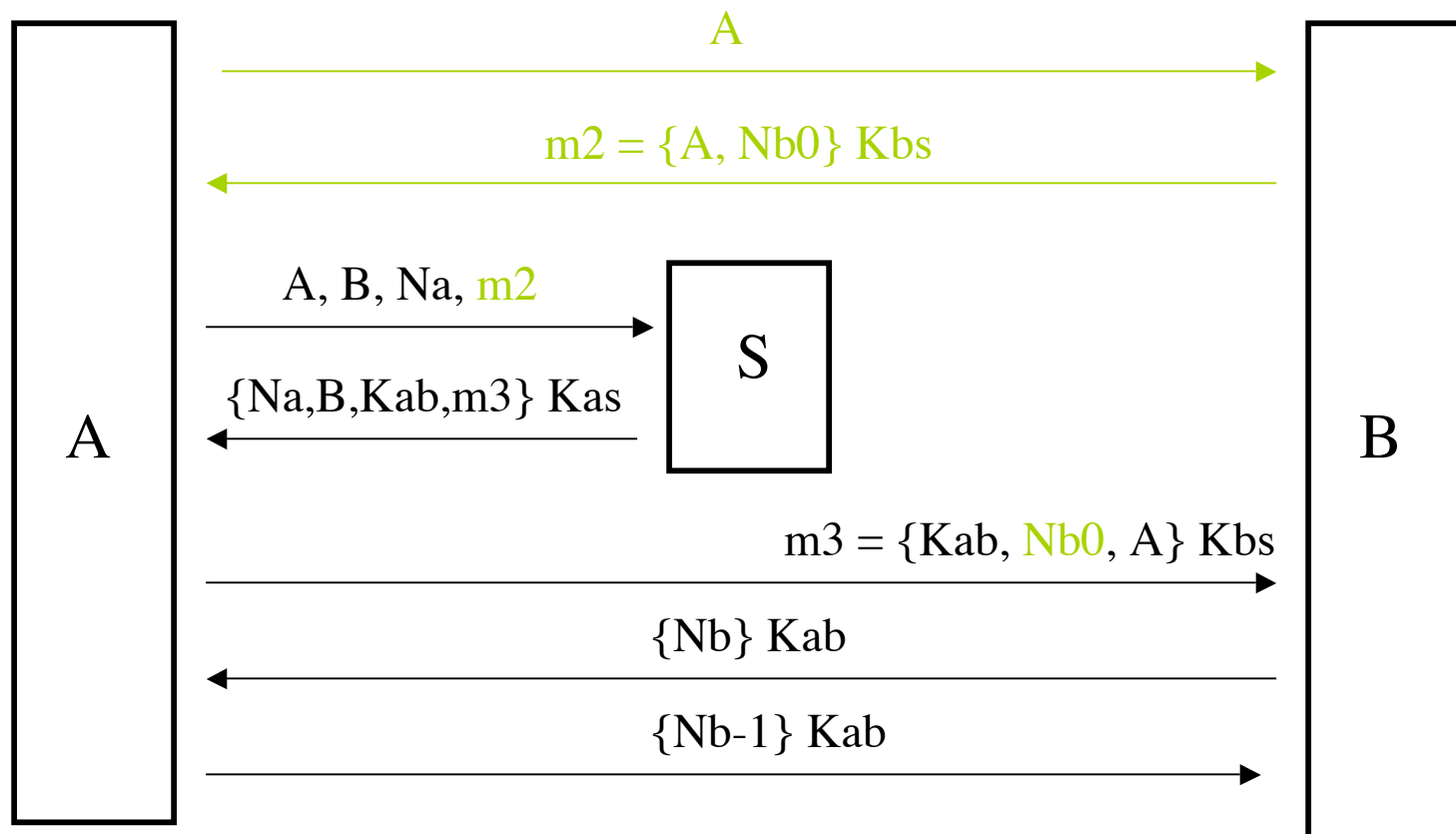
- ❑ Specify the toplevel functional requirements:
 - ❑ data transfer --> ftp
 - ❑ distributed data transfer --> bittorrent (!!)
 - ❑ medium access --> 802.x
 - ❑ authentication --> Needham-Schroeder, Otway-Ree's etc.
- ❑ Specifications drawn up directly from functional requirements.
 - ❑ State machine diagrams as a side note.
- ❑ Proof of correctness was ad hoc
 - ❑ Errors !
- ❑ Security protocols are especially tricky.

Needham Schroeder authentication



- Celebrated! Or at least best known!
- Possibility of a replay attack (on Kab being compromised).
- B has no way of verifying that $m3$ is in fact new.

Amended Needham Schroeder authentication



Jump to the last page...

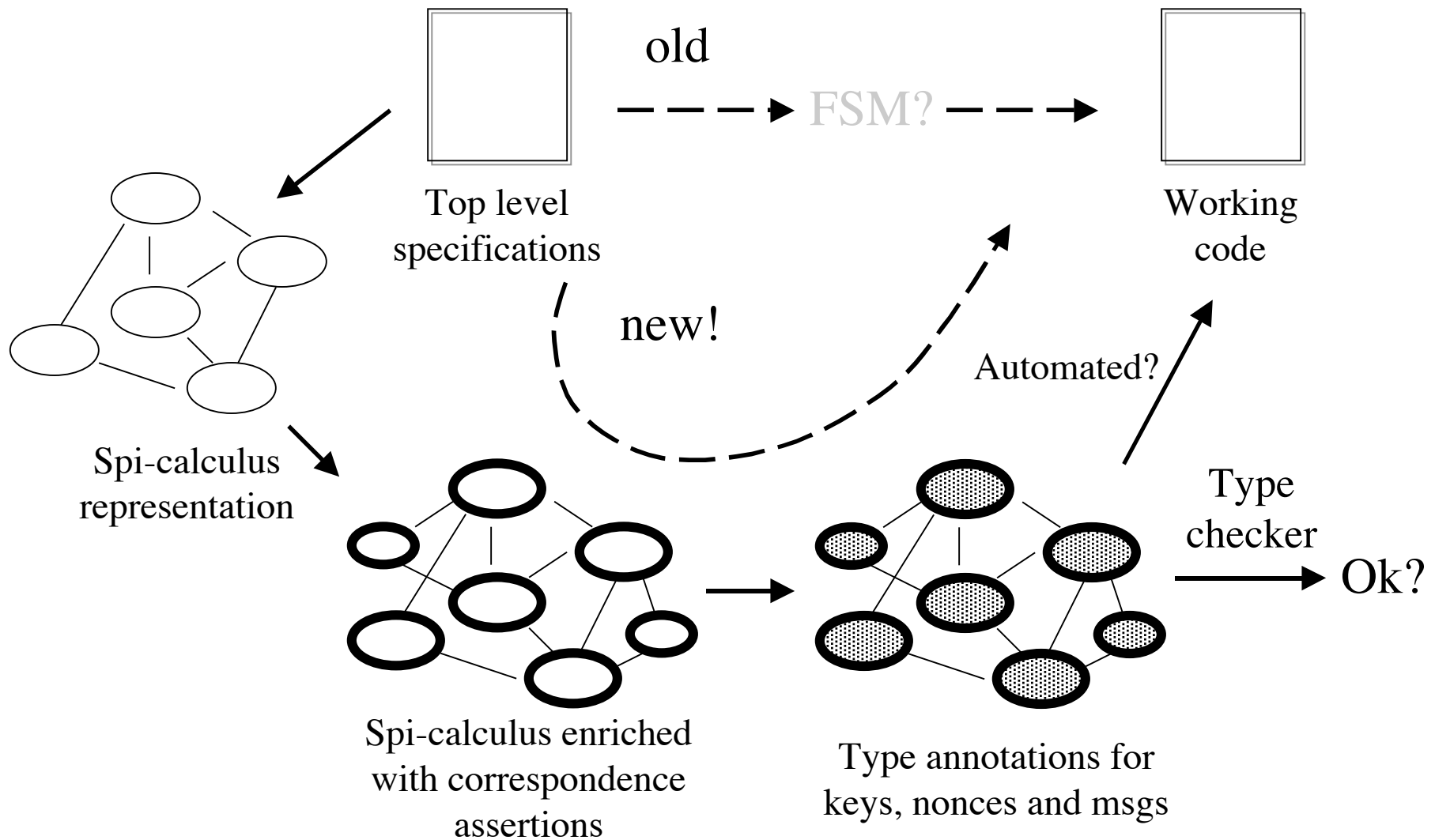
❑ Needham Schroeder ‘type-checked’:

```
$ java -jar ../../cryptyc.jar ns.cry
Error, rooted at line 134:
In end-statement: Can't guarantee that a correspondence has begun.
Required correspondence:
  begun(providing session key kab to b for a)
Available correspondences:
  None
```

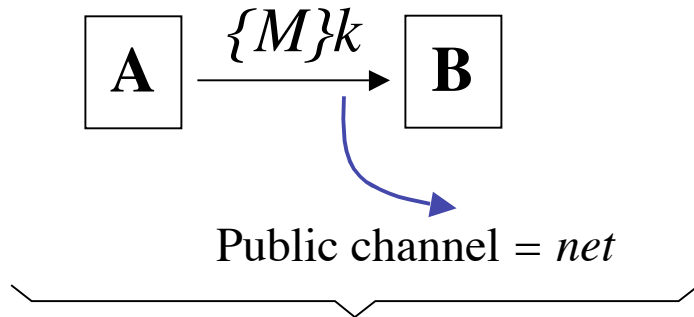
❑ Modified Needham Schroeder:

```
[poseidon]$ java -jar ../../cryptyc.jar ns-modified.cry
Type checked OK!
```

The overall view



Spi Calculus - by example

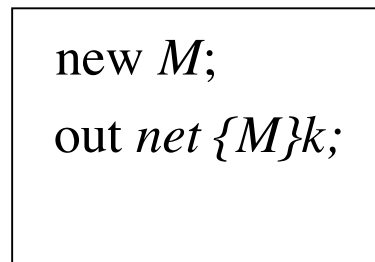


Names and Msgs:

- name: x
- pair: (M,N)
- empty: $()$
- tagged union

$\text{new } k; (A|B)$

repeat



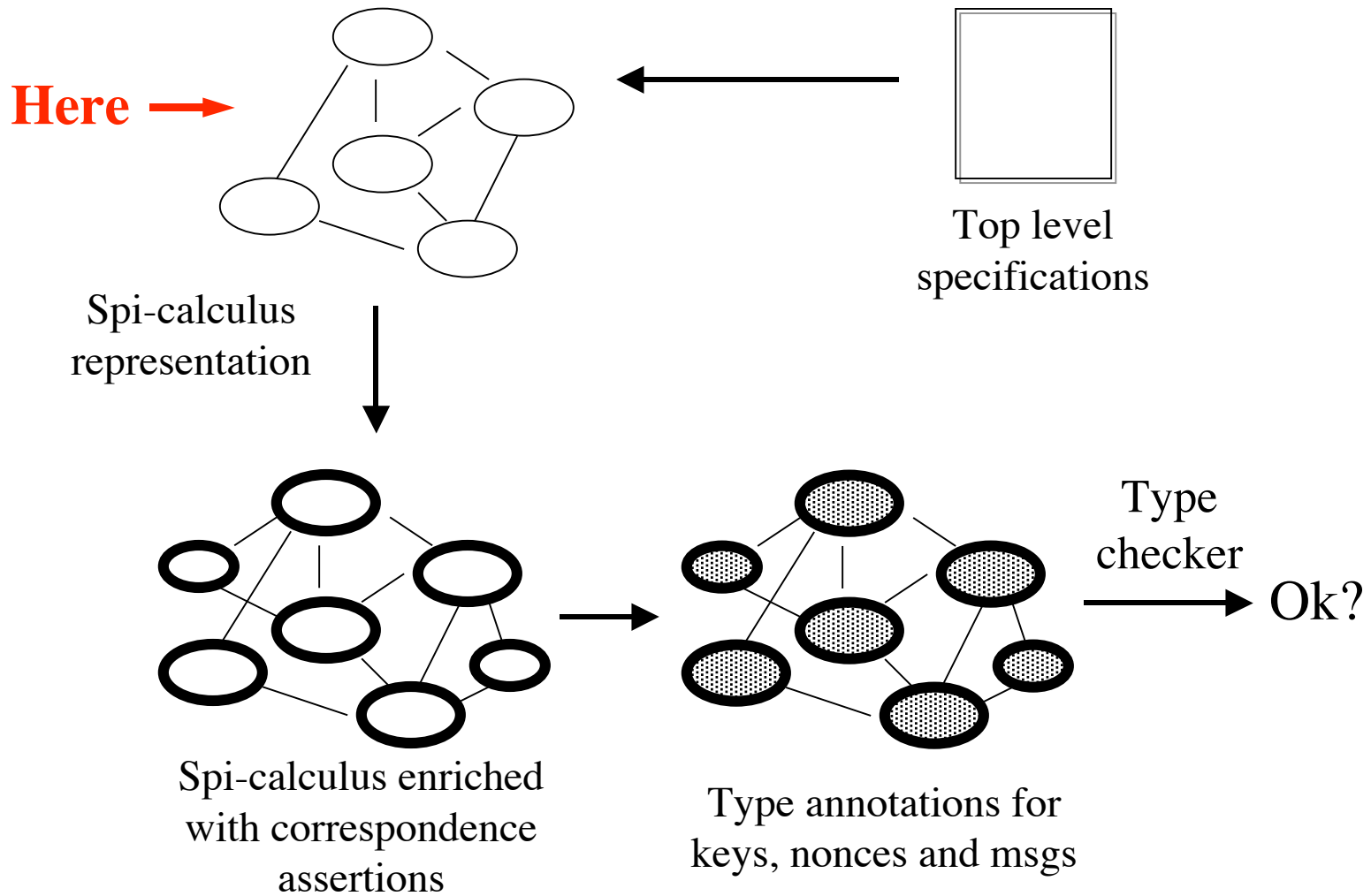
repeat



Processes:

- out $M N$
- inp $M (x:T);P$
- $P | Q$
- others...

Outline



Correspondence Assertions

- Augment the calculus:

- with new *events*:

- begin L

- end L

- with new *processes*:

- begin L; P

- end L; P

- Allows the correlation of events across processes

- Prevents *man-in-middle* and *replay* attacks

Eg:

- begin L; begin L; end L; end L

- begin L; end L

- begin L; begin L'; end L; end L'

- end L

- begin L; end L; end L

- (begin L | end L)

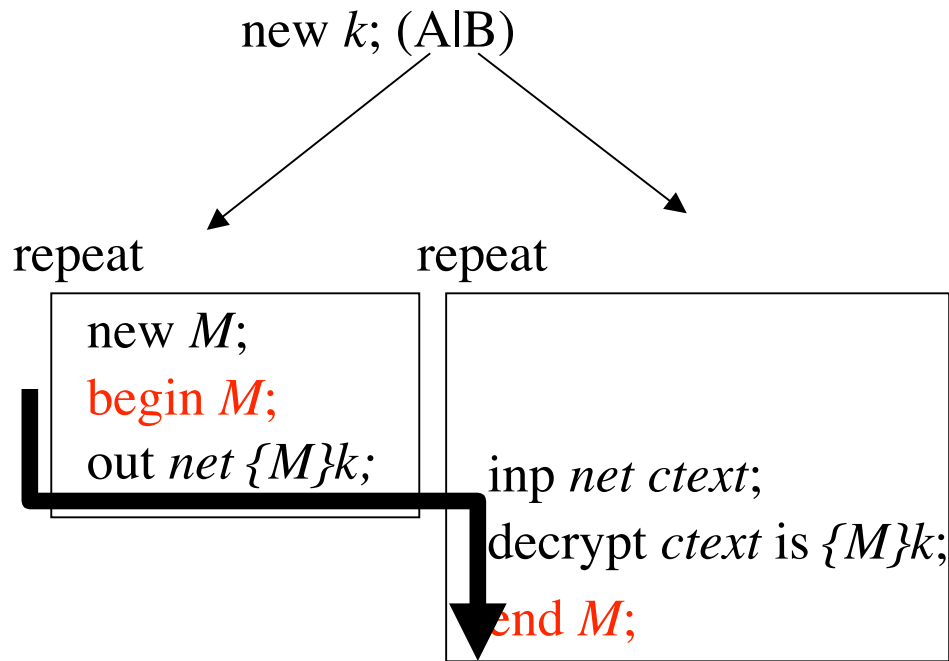
Safety and Robust safety

- *Safety*:
 - P is safe \Leftrightarrow each end L has a corresponding begin L
- Opponents: Arbitrary processes.
 - Untyped
 - Cannot assert events
- *Robust safety*:
 - P is safe \Leftrightarrow (PIO) is safe for any arbitrary O.

Note:

- Might seem vacuous but in fact is not
- Consider the opponent who does a replay attack:
inp *net msg*; out *net msg*; out *net msg*;

Example



- Solution: Add nonce!

Event 1: A begins M

Message 1: B->A: n

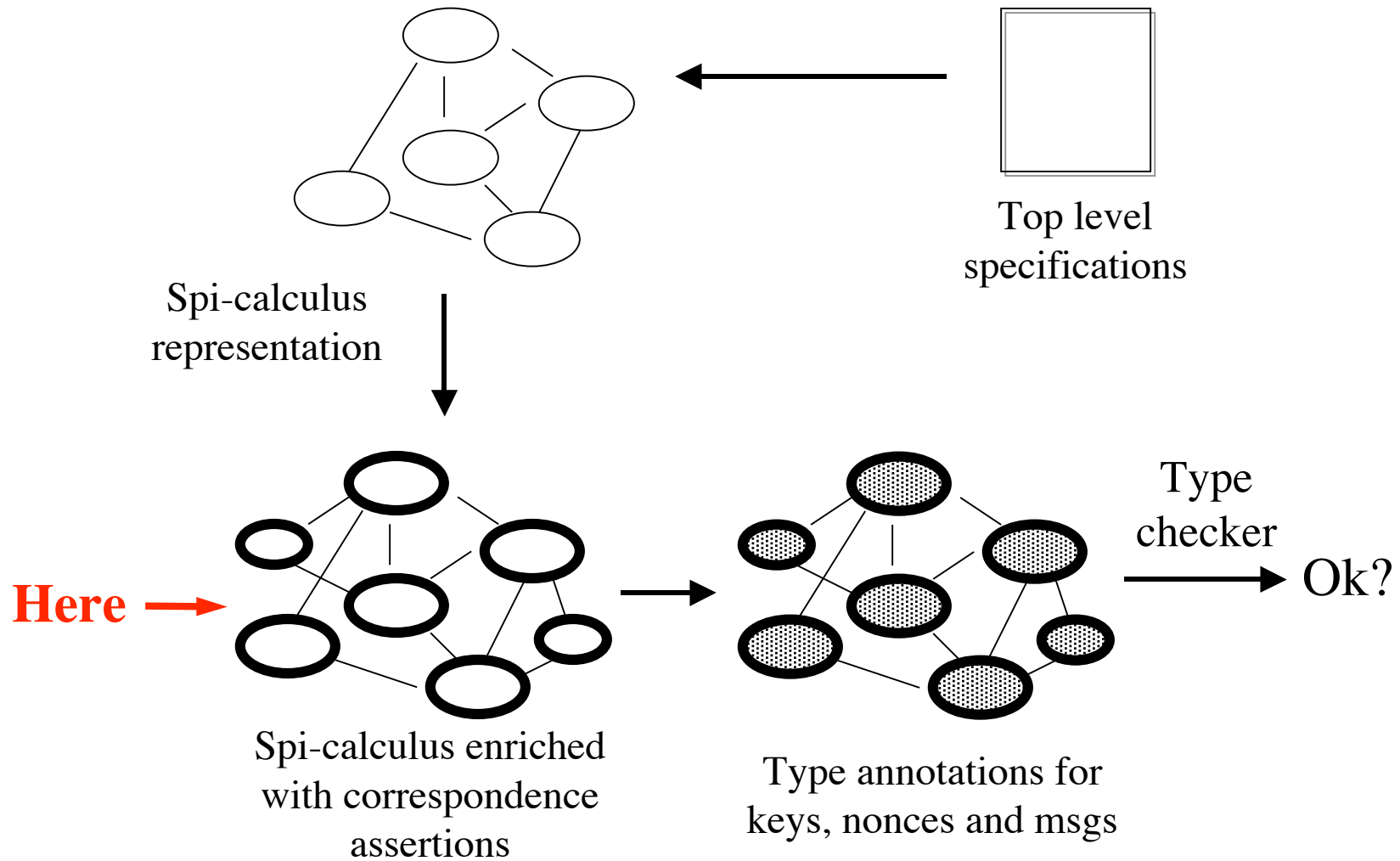
Message 2: A->B: $\{M,n\}_k$

Event 2: B ends M

- Now robustly safe!
- But need to check that:
 - *Type system!!!*

- Allows us to correlate across processes
- Opponent which breaks this:
 - inp net $ctext$; out net $ctext$; out net $ctext$
- Therefore safe but *not robustly safe*.

Outline - status check



Typing protocols

- ❑ Data types
 - ❑ **Un**: The untrusted type.
 - ❑ Public data and channels : anything exposed to the adversary
 - ❑ Trusted types:
 - ❑ $\text{Ch}(T)$: channel, $\text{Key}(T)$: key for encrypting type T etc
- ❑ Effects for processes
 - ❑ Keep track of unmatched end assertions.
 - ❑ Something like: P : [end L , end L , end M , end N ...]
 - ❑ Tracking sequential code is easy.
 - ❑ What about transferring effects over parallel processes?
 - ❑ The yellow arrow!
 - ❑ Some kind of temporal precedence needs to be established.

Nonce handshakes establish ordering!

- ❑ Paper's most relevant contribution (in my opinion)
- ❑ The *only* way to create a nonce is to *cast* it.
 - ❑ Cast N is $(x:\text{Nonce } effects);P$ ----- (cast is a new Process!!)
- ❑ A check primitive
 - ❑ Exists
 - ❑ If typechecked - proves that a preceding process called a cast
 - ❑ Processes now have effects of this form:
 - ❑ $P: [\text{end } L, \text{end } M, \dots \text{check } N, \text{check } Q]$
 - ❑ Allow the type checker to ensure that at 'new' nonce generation time , ie $(\text{new } N:\text{Un}; P)$, check N occurs at most once in the effect of P

An example

- Scenario:

- n is a nonce that is somehow already shared between the processes.
 $n:\text{Un}$
- c is a private channel (so can have type other than $\text{Ch}(\text{Un})$) - so that we do not have to bother about encryption and decryption of the nonce
- Specifically it has type $\text{Ch}(\text{Nonce} [\text{end } m])$

- $P =$

```
begin m;  
cast n is (n':Nonce [end m]);  
out c n'
```

- $Q =$

```
inp c (x:Nonce [end m]);  
check n is x;  
end m
```

- $R = \text{new } (n:\text{Un}); (P|Q)$ typechecks?

An example

Red: Processes

Brown: types

Blue: Effects - end/check

Green: Data

□ P =

begin m;

cast n is (n':Nonce [end m]);

out c n'

[end m]

□

P: []

Q: [check n]

R: [check n]-[check n] = []

□ Q =

inp c (x:Nonce [end m]);

check n is x;

end m

[end m]

[check n]

□ R = new (n:Un); (P|Q)

□ Rules (informally):

□ If $N:Un$ and $P:ef$

cast N is ($x:Nonce$ es); $P: (es+ef)$

□ If $N:Un$ and $N':Nonce$ es and $Q:ef$

check N is N' ; $Q: ((ef - es) + [check N])$

□ If $P:ef$

new (N); $P : (ef-[check N])$

Post mortem analysis!

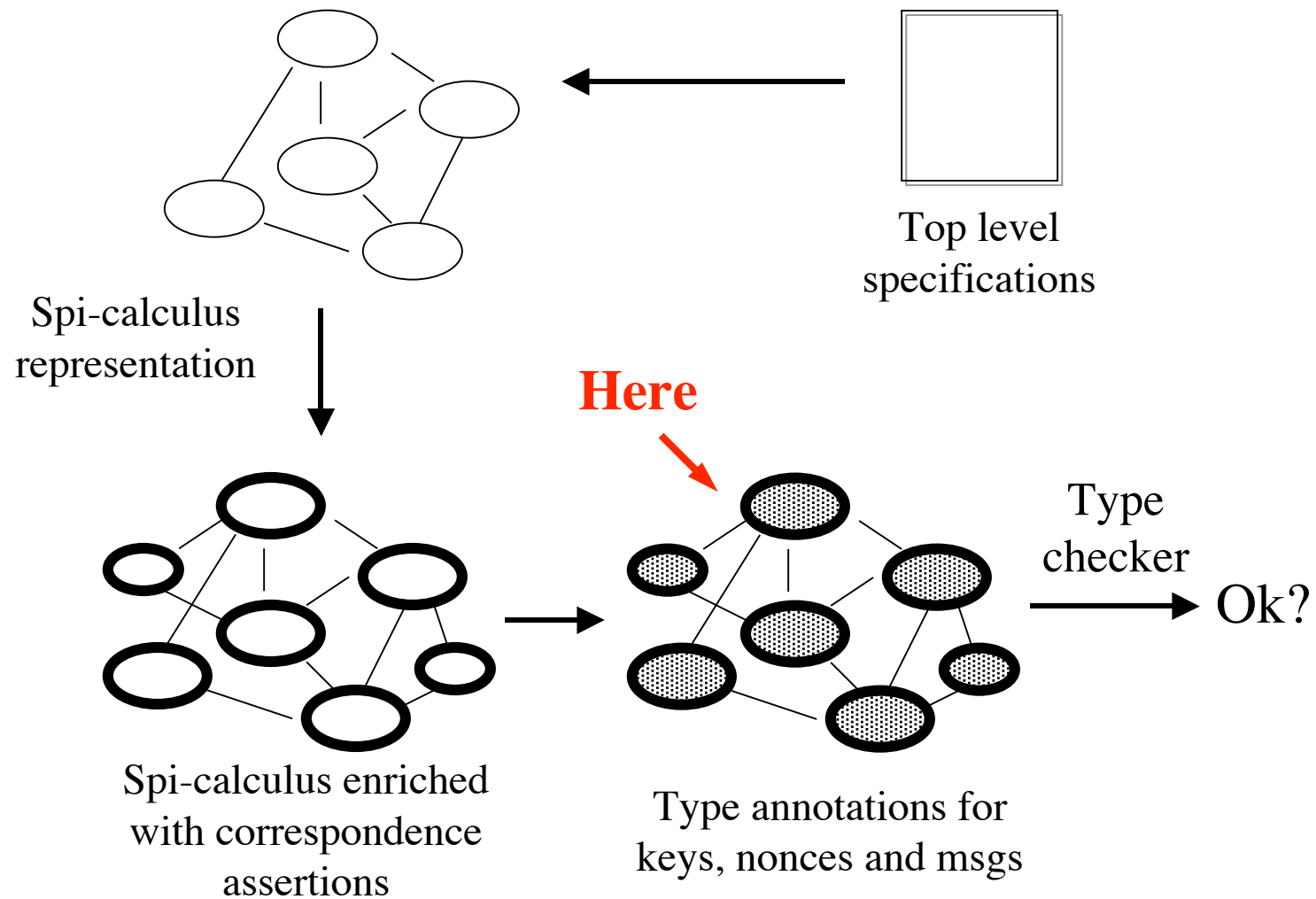
- ❑ Notice what happened.
- ❑ Cast:
 - ❑ The *cast* ‘encapsulated’ all the begins into the effect that nonce ‘carried’
 - ❑ Since a cast is the only way that a *nonce type* can be created => nonces will always carry effects to other processes
- ❑ Check:
 - ❑ A *check* causes the reverse at the receiver process
 - ❑ In addition it adds to the effect multiset an effect which ensures that one and only one *new* matches the check. (single use of nonces)

Conceptually:

```
begin a;  
begin b;  
....  
cast ____ es
```

```
check ____ es  
....  
end a;  
end b;
```

Outline - status check



A few comments about the type checking

- Code already annotated with types
- Deterministic choice of judgement rules for types (msgs and names) *and* effects (processes)
- For Un: everything type checks
- Interesting mention:
 - Msg Encrypt
 - Proc Decrypt

Summary

- ❑ Provides a way of validating the ‘model’ that will be implemented.
- ❑ Uses correspondence assertions to verify ‘control flow’ (Lam and Woo)
- ❑ Major contribution is the successful typing of nonce handshakes. Nonce handshakes allow processes to sync.
- ❑ Forms the basis of verifying correctness of public key based protocols

Criticism

- ❑ Model does not handle timestamp based protocols. Although it does not seem hard to incorporate.
- ❑ It is programming of the protocol! And that too with hard restrictions on the types.
 - ❑ Prone to error? - but probably less than manual spec.