Fraunhofer Institut für Offene Kommunikationssysteme

Policy Group Control Issues

Michael Smirnov
Fraunhofer FOKUS

27.-31.10.2002, Dagstuhl, Seminar Nº 02441
Quality of Service in Networks and Distributed Systems

What we want and what we need for this

- Open
  - Standard rules for service syntax and semantics
    - → interoperability
    - → portability
- Flexible
  - Easy to configure a service out of components, add/delete components from different vendors
    - → extensibility
    - ← interface definitions between components
- Scalable
  - → design for scalability evolvability

Policy

- Why to change components?
  - To provide optimal policy for particular user/application (i.e. for a Subject)
- What is policy?
  - *„Policy is a rule that defines a choice in the behaviour of a system“* [M. Sloman]
  - Component:= policy | mechanism
- How to separate concerns?
  - Subject → PolicyAgent → TargetObject
  - PolicyAgent(P_1, ..., P_N)
  - TargetObject(a_1, ..., a_M)

Fraunhofer Institut für Offene Kommunikationssysteme

Further separation of concerns

- Separation of concerns between obligation and authorisation
  - {Obligation; Authorisation} $\times$ {Positive, Negative}
    - S → A+ → T(a_i): subject may request action a_i on T
    - S → A- → T(a_i): subject may not request action a_i on T
    - S → O+ → T(a_i): subject must request action a_i on T
    - S → O- → T(a_i): subject must not request action a_i on T

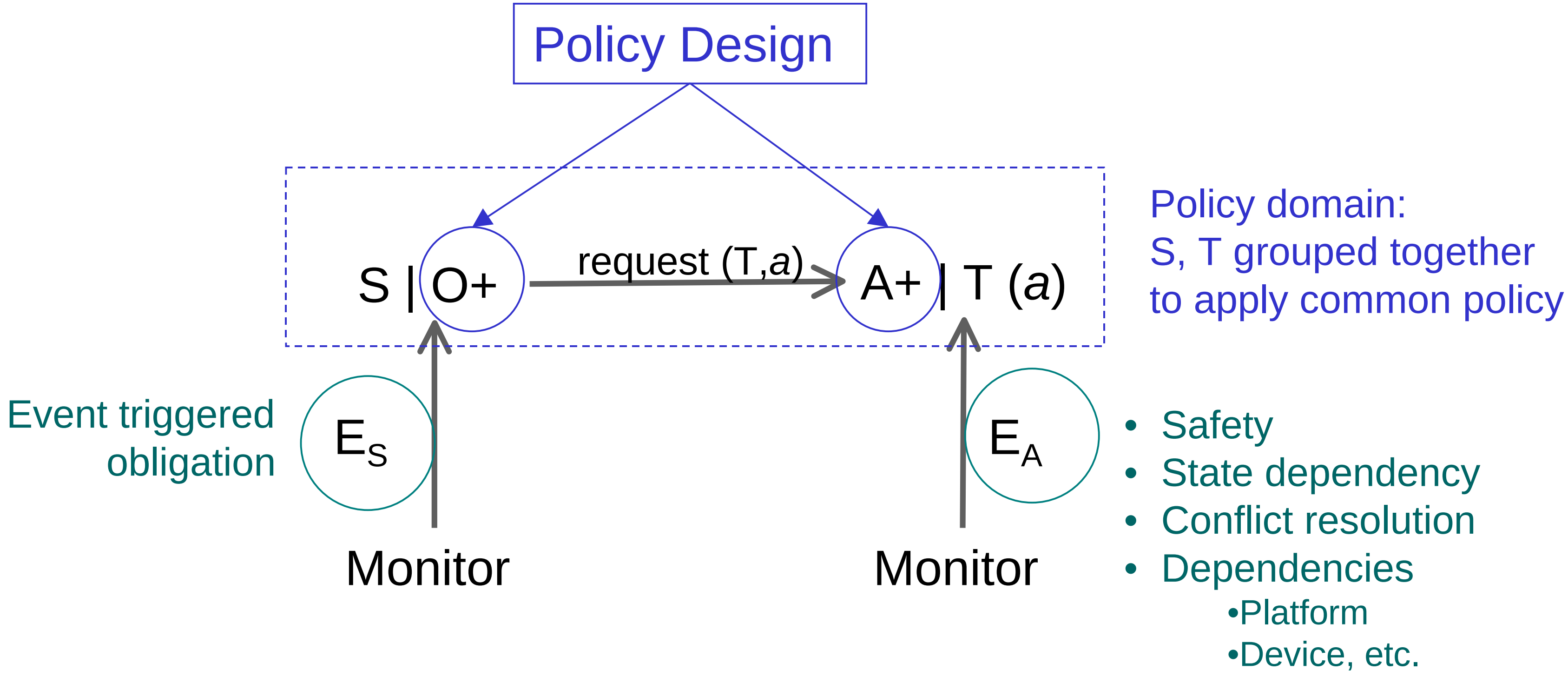
	Entity type	Relation	Configured policy	Discovered policy	Purpose of Conf. policy
T	object	server	A	O	safeguard
S	role	client	O	A	behaviour

- O-policies are S-based, A-policies are T-based:

S | O → A | T

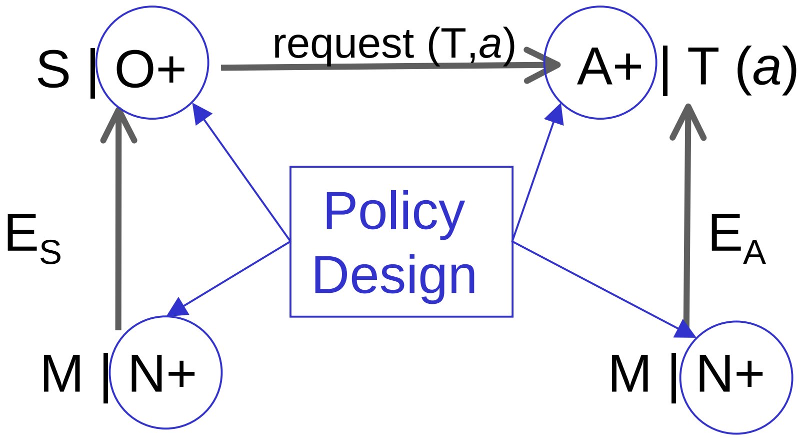
PolicyAgent disappears

Closed loop control



- Future Directions: „dynamically change behaviour to cater for new services“ [M. Sloman]

1st step: Monitoring behaviour



- Notification policy ~ Obligation for notification
  - M | N+ → E_S → S | O+

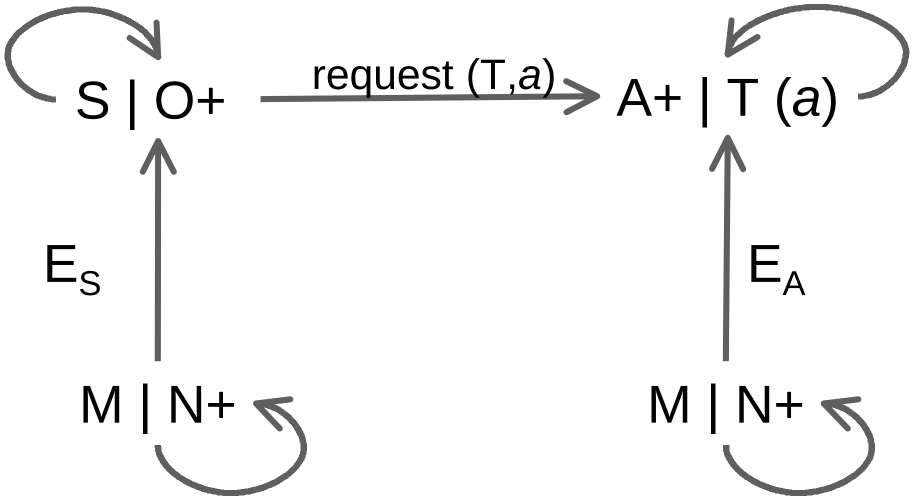
Co-ordinated policy set design

- Composite service: → service system (group)
  - more than one type of target objects under control of potentially more than one Subject (Manager)
- Scenario based design:
  - Scenario is {understood | feasible | ... } instance of service implementation in a given infrastructure
  - Scenario = <S, T, O, A, E>
- Service policy:
  - Evolving concept
  - Set of all implemented scenario policies

Why scenarios?

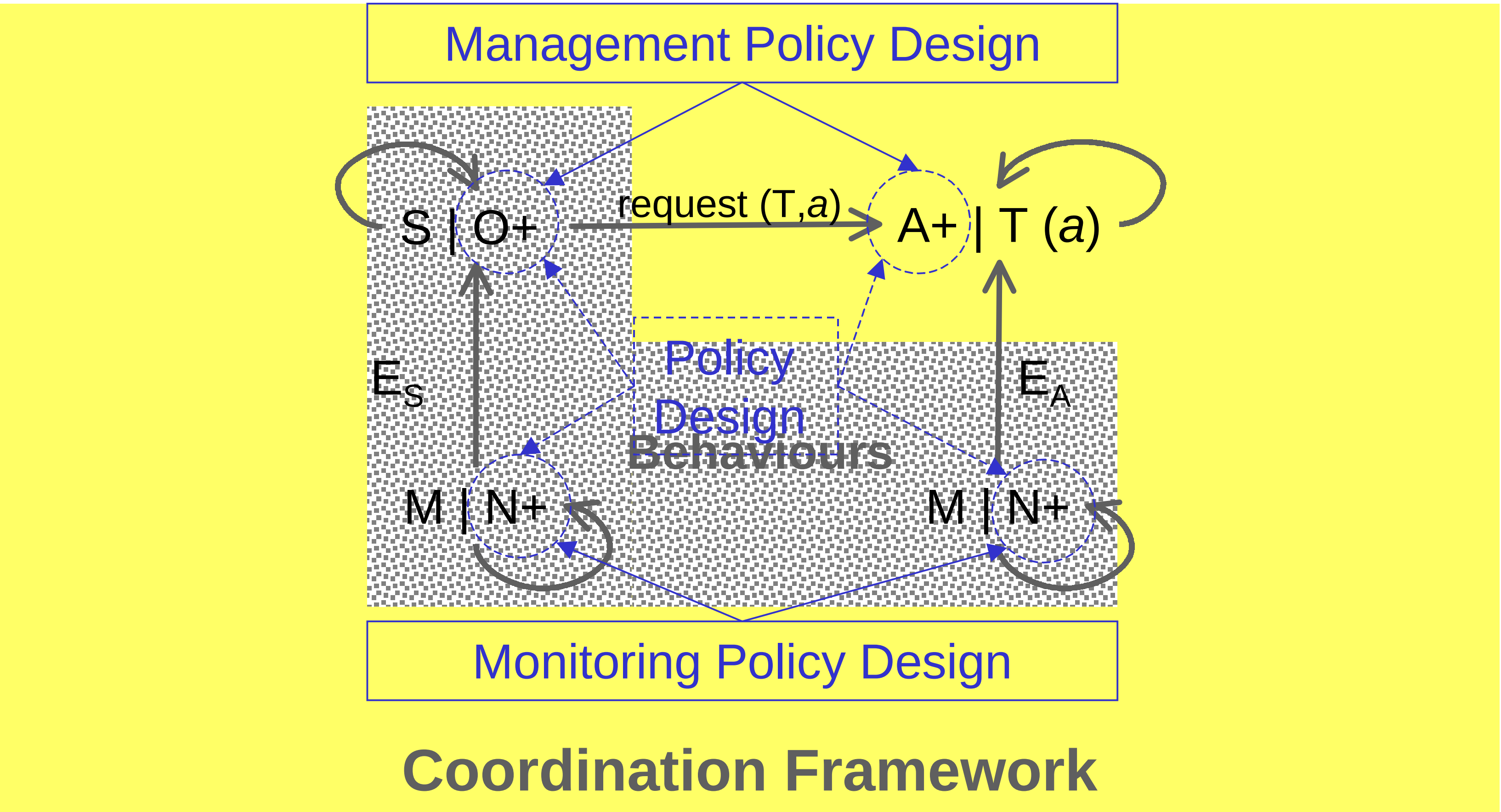
- Service definition is hard
  - Is BE IPTel the same as EF IPTel?
  - Is cached service the same as not cached service?
- Scenario is a good thing
  - Incremental service deployment
  - Re-use of components
  - Scenario based design is a natural way of design for evolvability (handle tussles):
    ExistingService.S_1, S_2, ..., S_I, ..., S_N, ...(NewService)

GENeralisation: Internal Events



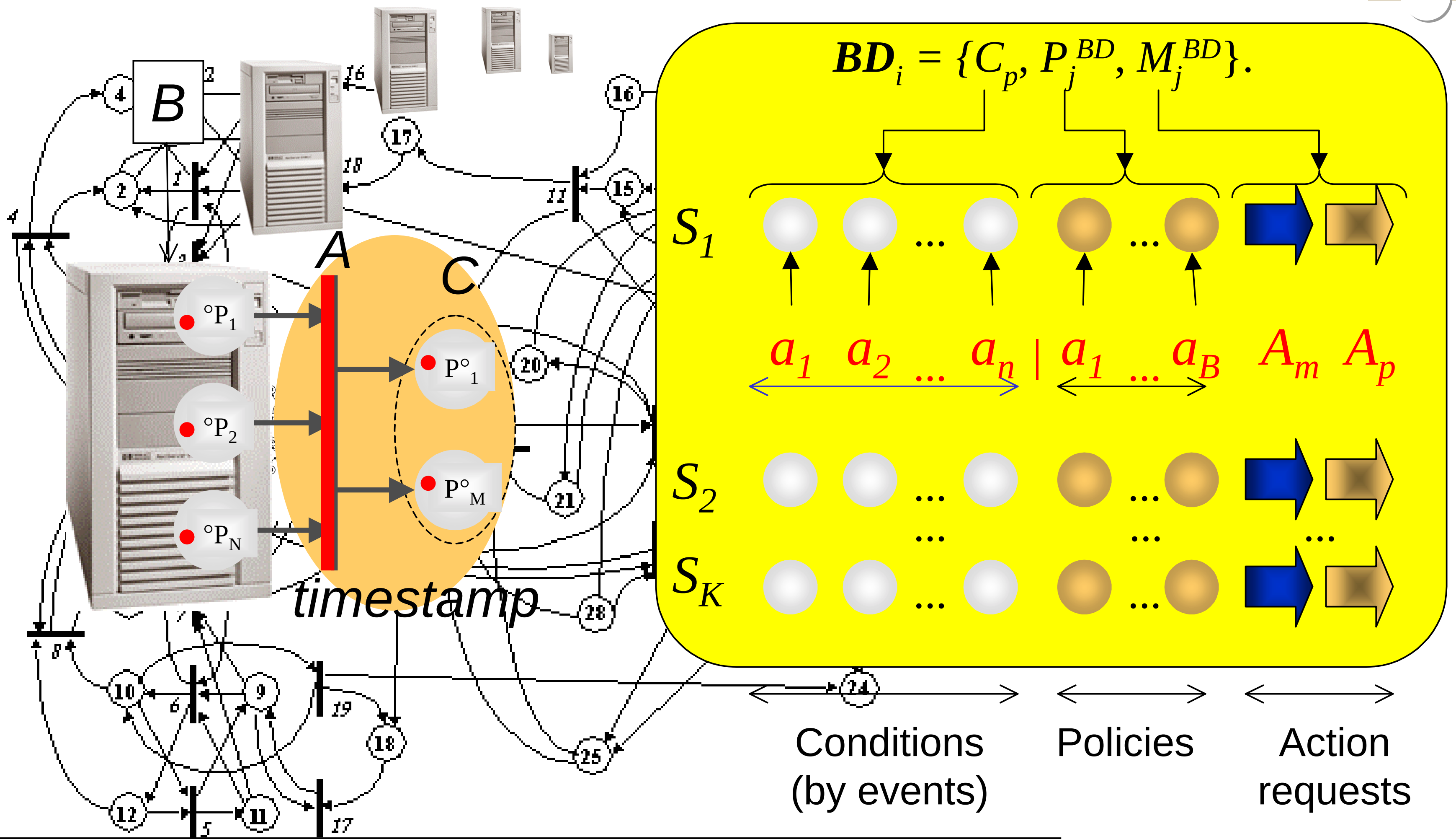
- Internal events (state) at Subjects, Objects, Monitors justify their roles in service system as *event correlation points* (mediators)
- One man‘s internal event is another man‘s external event → [service] group event notification

GENeralisation: Multiple Designs



- Multiple behaviour designs are inevitable → need coordination framework for semantics, trust, syntax

GEN: Group Event Networking



Event = {**A**ction, **B**ox, **C**onditions, **D**uration)

Example 1: Service Creation

oneof (exist(v1) AND exist(s1)) Pr1

RS1

```
Start & Next1 & C → play(v1);
v1_fin            → send(v1_fin);
HEARD(v1_fin)     → play(s1);
HEARD(v2_fin)     → play(s1);
HEARD(v3_fin)     → play(s1);
HEARD(Stop)       → Stop
```

oneof (exist(v2) AND exist(s2)) Pr2

RS2

```
Start & Next1 & C → play(s2);
HEARD(v1_fin)     → play(v2);
v2_fin            → send(v2_fin);
HEARD(v2_fin)     → play(s2);
HEARD(v3_fin)     → play(s2);
HEARD(Stop)       → Stop
```

oneof (exist(v3) AND exist(s3)) Pr3

RS3

```
Start & Next1 & C → play(s3);
HEARD(v1_fin)     → play(s3);
HEARD(v2_fin)     → play(v3);
v3_fin            → send(v3_fin);
HEARD(v3_fin)     → play(s3);
HEARD(STOP)       → Stop
```

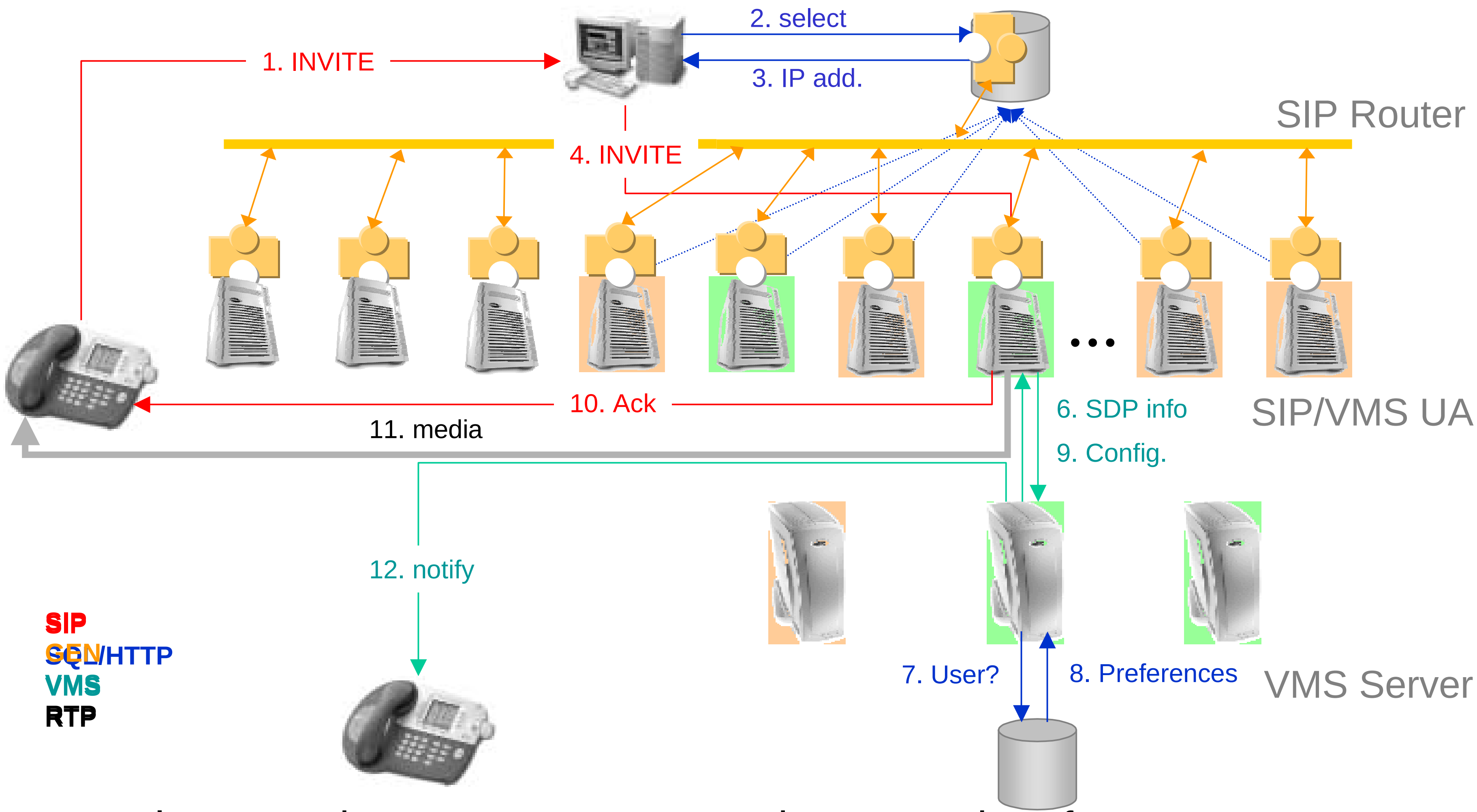
oneof (exist(v4) AND exist(s4)) Pr4

RS4

```
Start & Next1 & C → play(s4);
HEARD(v1_fin)     → play(s4);
HEARD(v2_fin)     → play(s4);
HEARD(v3_fin)     → play(v4);
v4_fin            → send(Stop);
HEARD(Stop)       → Stop
```


Fraunhofer Institut für Offene Kommunikationssysteme

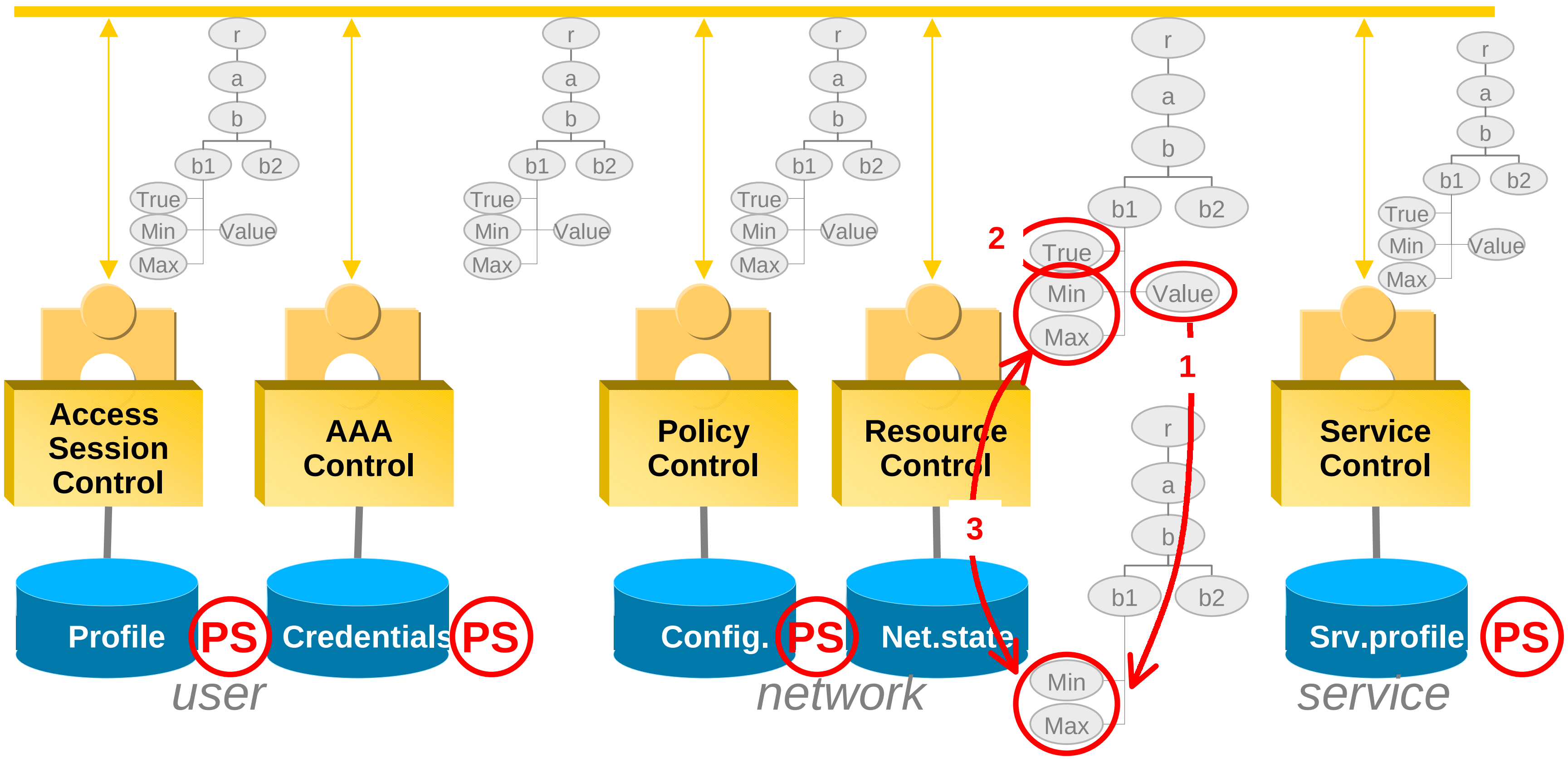
Example 2: Load Balancing



- It scales! No reboots, no re-programming, same interfaces
- Failures are treated as naturally as busy state

E.G. 3: Conflict free policy computation

- Conformance to SLA → service ontology (practically: a tree)
- Negotiatable parameters →meta-data (practically: modality+range)



- Group communication (Partial State) →up to 70% latency reduction

Conclusion

- „What is envisioned is a network of unmanned digital switches implementing a self-learning policy at each node so that overall traffic is effectively routed in a changing environment--without need for a central and possibly vulnerable control point“
- „The network can be made rapidly responsive to the effects of destruction, repair, and transmission fades by a slight modification of the rules for computing the values on the handover number table“

Source: Paul Baran ODC, **1964**, v.1. RM4320, ch4