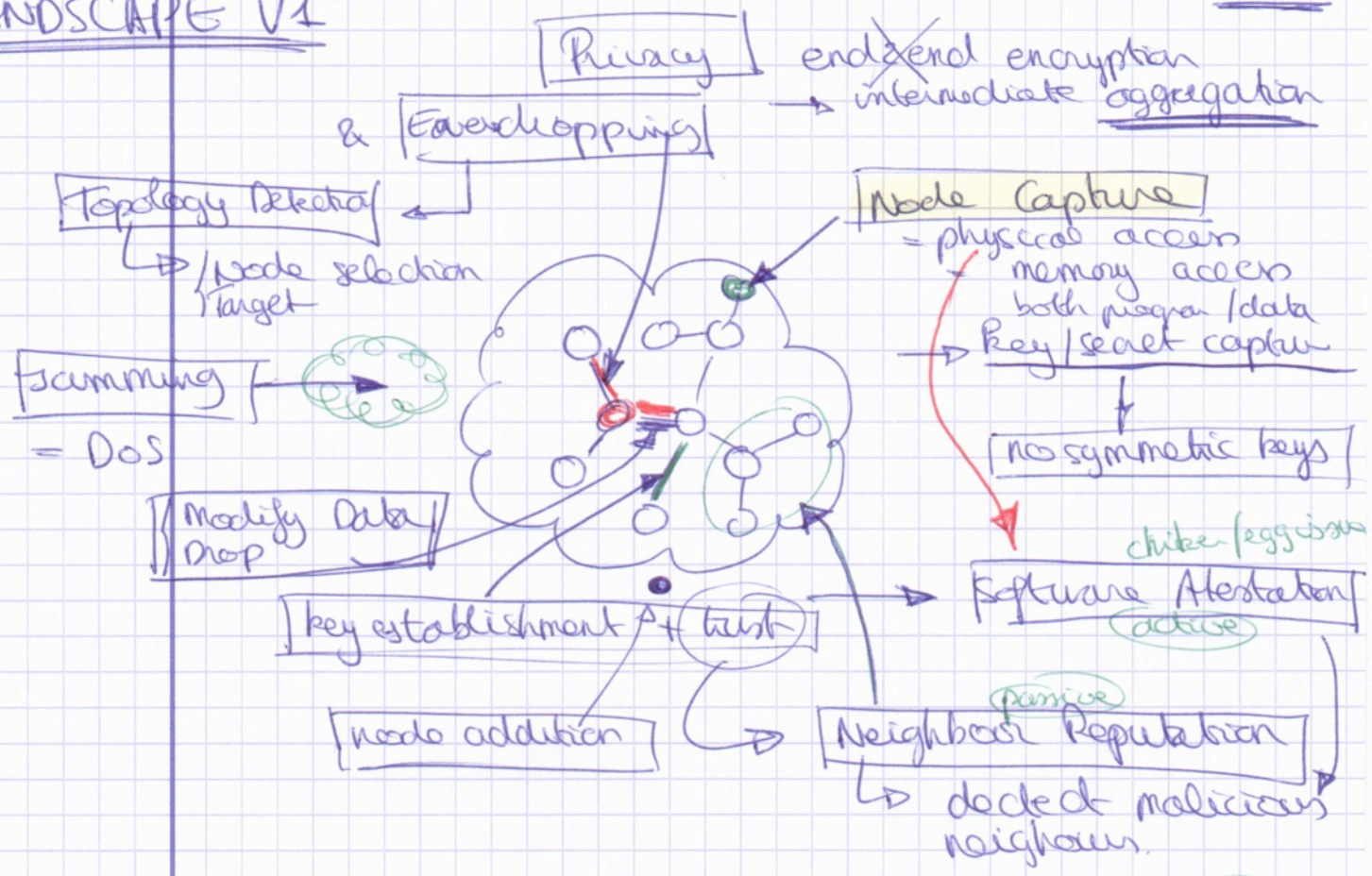
notes 1

LANDSCAPE V1



NOW active Intrusion → physical access to 1 node (HIDS) (local detection)

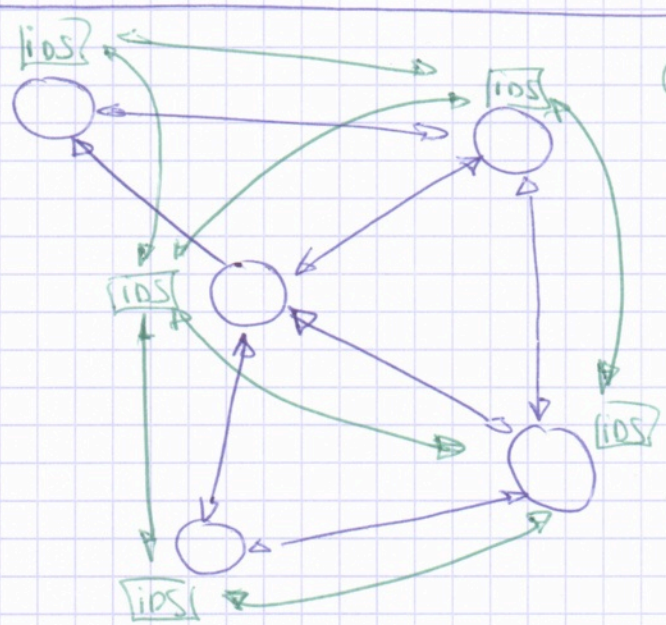
PAST re-active → compromise of 1 node in network. ← network detection (GIDS) / anomaly detection

FUTURE pro-active → over the wire access (NIDS)

+ node addition

! intelligent agent

! trust agents

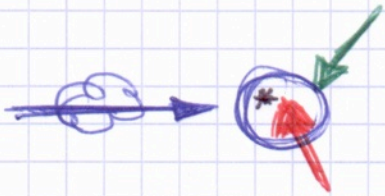


IDS overlays network.

agent – local
– remote exchange

CONCEPT V1

Notes &



FW

accept all kinds

- Network attacks
- Host attacks (~ physical)
- App attacks → very specific
- postmortem " → buffer overflow & CO
  ↳ it is done (SA) ↳ also brute force auth.

① event detection
② event correlation → anomaly detection
③ ↳ attack detection.
④ " reporting.

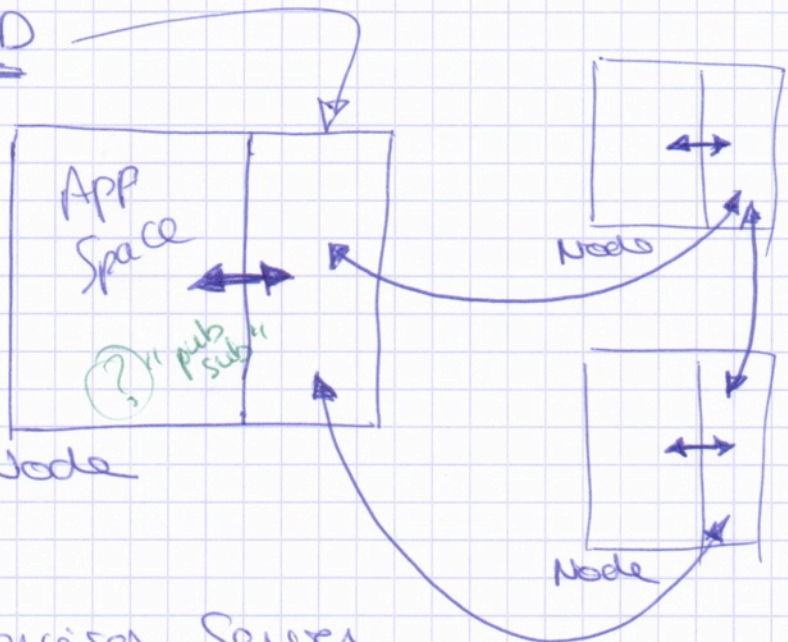
Node → | Network Group → server

LooCID

Loosely Coupled Intrusion Detection

:-)

≈ overlay network

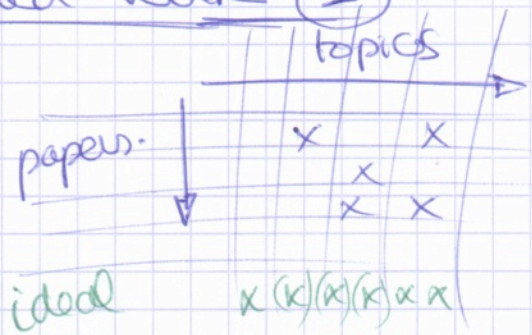


all communications are events.

FW (small)!! ← network proc
- comm
- auth
- pubsub
- correlate
- support all

+ Supervisor Server
↳ origin of "rules" | policy
↳ pushes through network
→ dynamic / group specific
(?) ↳ realtime response?
↳ to augment level
↳ divide network among special nodes

Matrix Related Work (1)



Thesis Structuur

Voorwoord / Dankwoord.
Samenvatting
Inleiding.
- Draadloze sensornetwerken
- Toepassingen ← VB !!
- Probleemstelling
- Doelstelling
- Verloop / Structuur tekst

Achtergrond.
- ~~...~~ (landscape, "notes" network, context, (cooci))
- Gerelateerd onderzoek
  - } major slices
  -

Probleemstelling ← Scenario's

2) Voorgestelde oplossing

Bestaande Technologieën
2
Architectuur
Implementatie
Discussie
Besluit

Demo

+ topology + supervisor

"simulate" 3 attacks ≤ 3 types

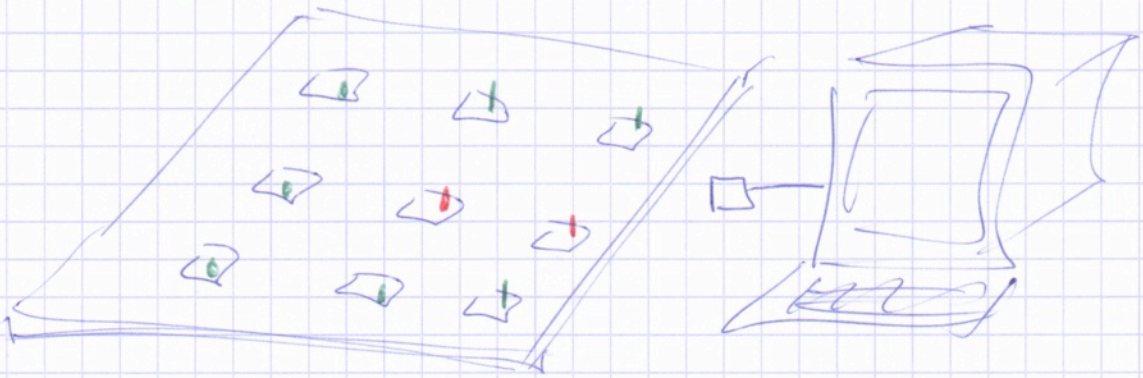
not implementable

network? application?

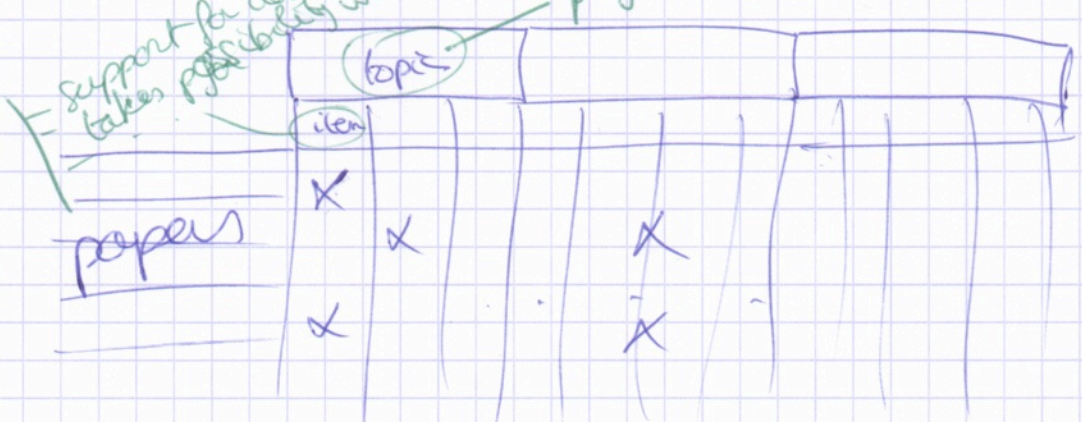
mind app? → with external interface

→ proximity? → marker detection

case "try not to be seen"



Malax related work (2) e.g. "physical"



Features

- end 2 end solution
- light weight
- non-intrusive (pub sub, event, ...)

report → Nodes ↓

node group ← GW - server

configure/install

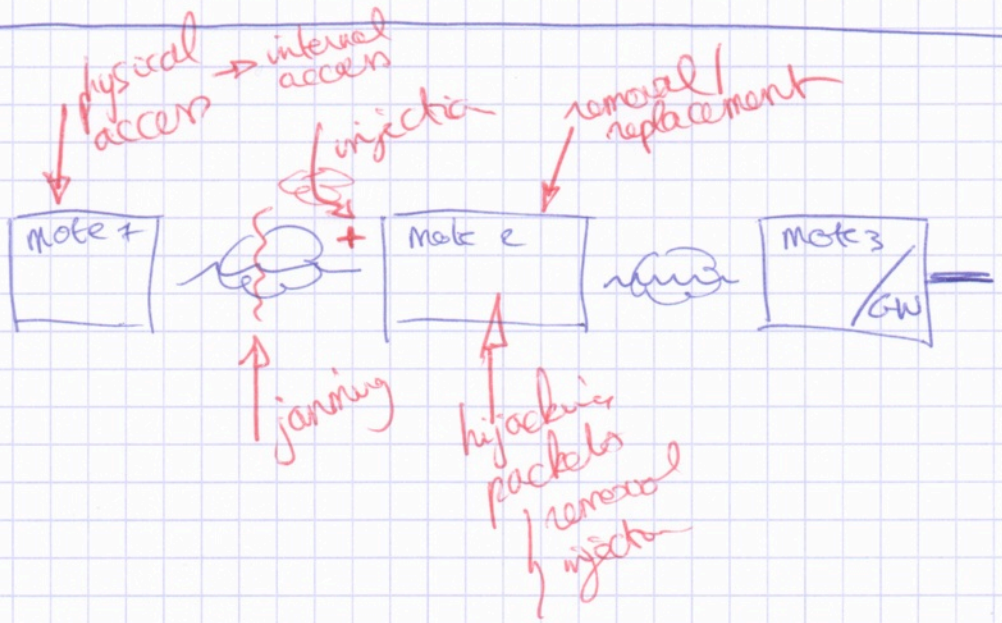
Reasons

- not in use — not in scope
  ↳ no offering

↳ detection while "sleeping" ? ...

↳ thesis NOT about detection = too large specific related work

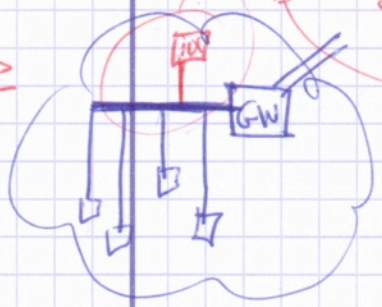
---



Notes 6

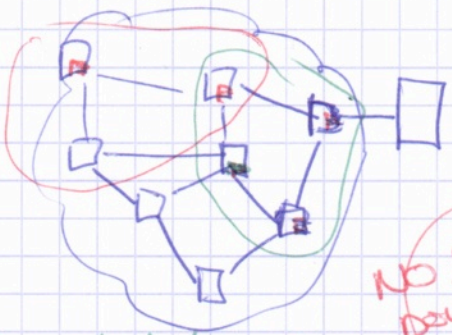
Differences "classic" IDS vs NSN-IDS

NIDS



"backbone" structure
see-all

vs

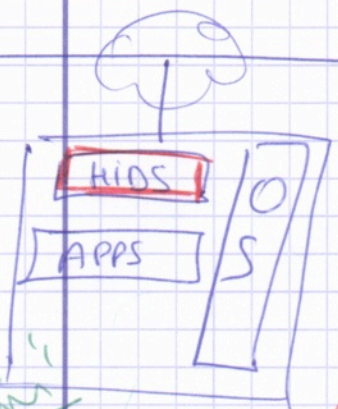


no single point of see-all

limited scope vision

=> no single instance possible

HIDS



"microsystem"

signatures

large list of events.

operating system "calls"

application specific

files focus. changes.

- END Node - ROUTER - coordinator
- Sensor Node - ClusterHead - sink

(simplifying / unified / centralized / generic library / DSL hides system details)

- request system info (local) → HOW? to abstract! → app specific info → memory addresses
- basic i/o "send"/"receive" → protocol "bytes"
  - ↳ between nodes
    - directive
    - informational → give request
    - remote "call"
  - ↳ listen promiscuous → "costly"
    - ↳ "a packet"? or (abstracted data format)
      - DSL access
      - optimized mem storage / cycles
- (K/V store) (structured byte array in mem) + offset, constants → (for computations) → reuse of code, "lighter" api (generic)
  - ↳ optimized storage with high level API
    - ↳ + fast increment/decrement
    - + event based "rule engine"
      - e.g. alert when counter x reaches y

DSL at dev time
↳ compiles optimal runtime code → OS - ignorant / independent → optimization.

↓

(POC) (test) of access hardcoded byte array is faster/smaller than "classic" code with nice interface.

↳ need literature ← example?! ← (DSL more optimal code than rewriting ??)

DSL → 1/ translate (compile) to "optimized | size / cpu" C-code } MoG (possible improved) "optimality" vs. classic API MoG.

@ gateway / controller / supervisor

2/ compile to (hex)

3/ OTAP onto nodes, according to policy

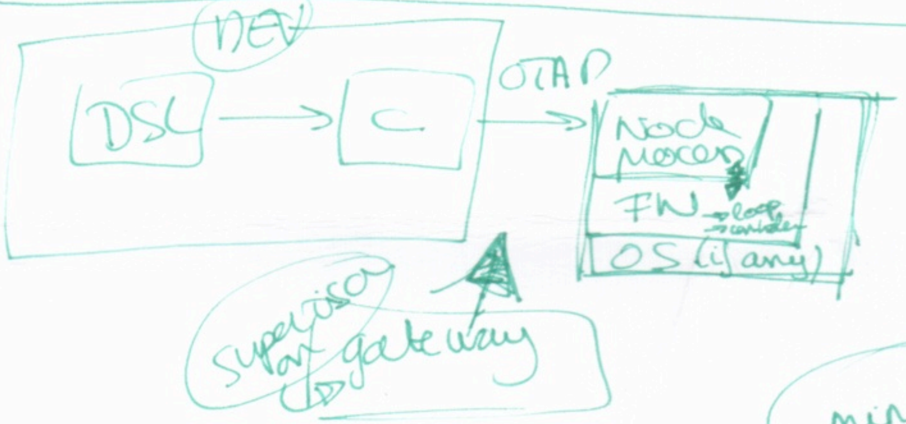
→ node-capabilities { based / dependent

---

optimize multi rules (in classic code this would be a no-go ⇒ imposed language..)

↳ functional

DSL → (language rewriting)+ → C... → direct to ASM? ≈ out-do compiler :-) ← avoid costly type checks?



"classic signatures" become ↳ DSL { pattern / anomaly / directive

↓ event based

minimal cmd structure

no loops!

1/ counterX ++

2) when counterX > 10

"actions" commands

1/ condition

2/ assignment (operator) command.

3/ event trigger definition

4/5/ ~~request~~ set/get 'info' → local system / remote / computed / messaging ↳ implicit

iDSL ~~industrial detection specification language :-)~~ cisco = ISDN SC

node { shouldn't / doesn't } know about others → "might not be there" ← abstracted

↳ check if possible to implement algo's that DO access specific other nodes' info.

notes 9

- DSL
- code generation } show possibility justify
- FW + API → very low-level technical.

LooCiD FW

Loosely Coupled Intrusion Detection FW

- Contiki (+ LooCi)
  - "or not"

(?) event driven ? DSL

(tooling) just may not be optimal path

thesis "scope"

subset possibilities → no DSL

just enough to prove overall possibilities

based on "detection"

literature →

1) detection WSN
2) detection classic.
3) DSL → code generation

Demo // show classic implementation
X 2-3 detections
versus (generated) low-level API + FW
show workings
+ code/mem/cpu reduction

attach selection → "visibility" ① "Sybil" ?
② "sink hole" ?
la mama ③ "node capture" ?
anomalie.

:->

notes 10/14

! limit max size = compiler policy

mem = compromise to CPU

hidden through DSL.

choose.

= battery

use more = speed = more lifetime

(reuse space)

reserved mem block

e.g. cyclic buffer size

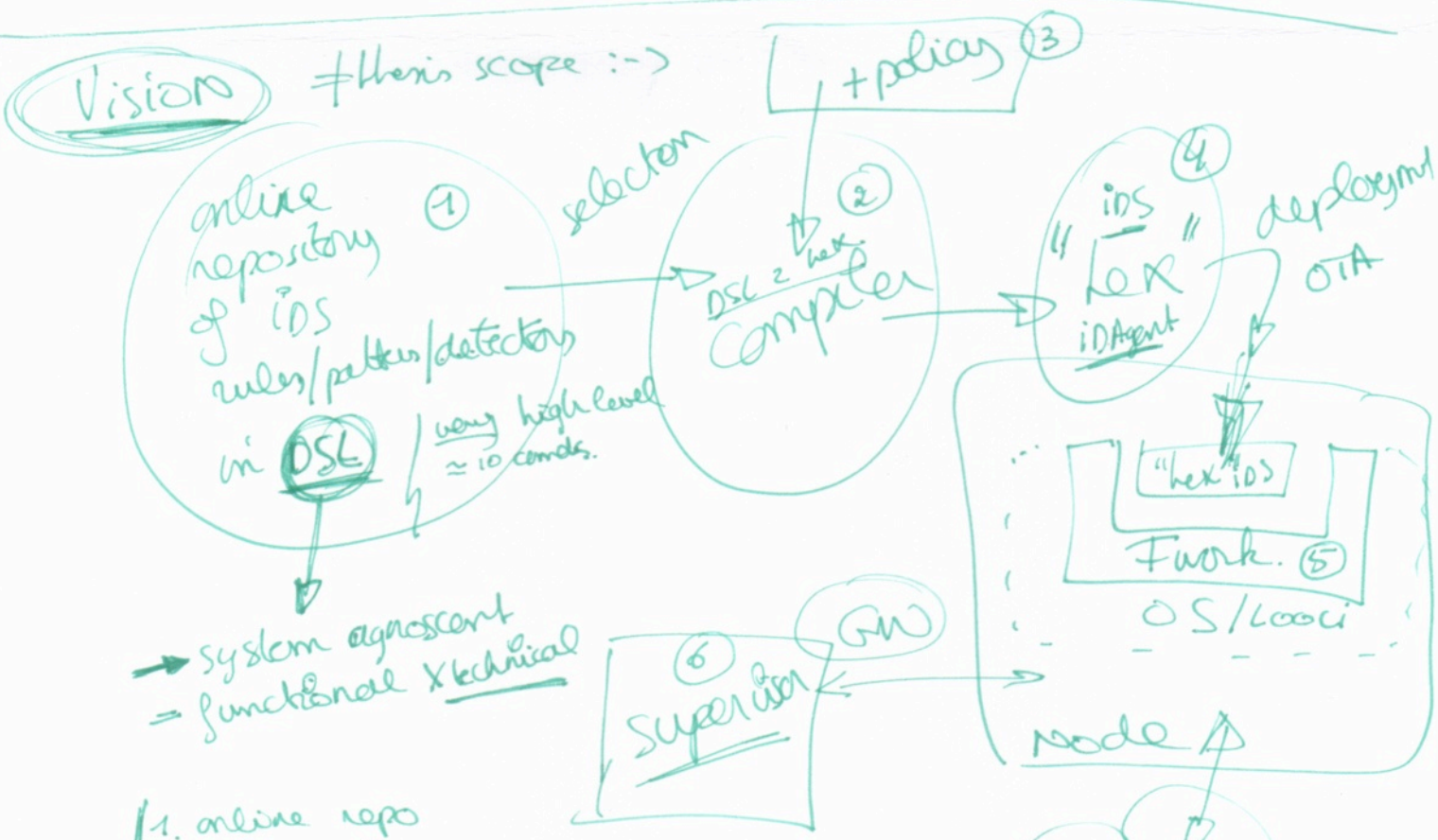
"transparent" ≈ conversion (functions)

↳ "encrypt" ?+

↳ subblock exchange.

possibilities show/pray X implement all ?

Vision ≠ this is scope :->



→ system agnoscent

= functional X technical

1. online repo
2. DSL compiler
3. policy
4. low level API implementation.
5. supported API framework
6. supervision tools

+ scope of Hardware → ATMEGA1284P + Contiki + Loosi

notes 11

POC 1

① RFSN

$T_{ij} = E(R_{ij})$

Architecture: watchdog → reputation → trust → behavior

second hand info

$B_{ij} <> TH$

actions other nodes

cooperative / uncooperative

WM Routing 1) promiscuous mode listen forwarding behavior

WM Data 2) outlier detection.

$= D_{ij}$

R_{ij} = rep of j @ i

= prob. distrib.

RT_i = all reps @ i

$R_{ij} = f(D_{ij}, R_{ij})$

recursive update

speed up convergence

$$R_{ij} = (R_{ij})_D + (R_{ij})_{iD}$$

$$(R_{ij})_D = f(D_{ij}, (R_{ij})_D)$$

$$(R_{ij})_{iD} = (R_{ij})_{iD} + w_{ik} * R_{kj}$$

$$= g(R_{ik})$$

⇒ BRSN

$$R_{ij} = \frac{P(D_{ij} / R_{ij}) * R_{ij}}{\sum P(D_{ij} / R_{ij}) * R_{ij}}$$

$$R_{ij} = \text{Beta}(\alpha_i + 1, \beta_i + 1)$$

not cooperative

Gamma $\Gamma(x) = (x-1)!$

Beta dist $P(x) = f(x, \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$

$X \sim Beta(\alpha, \beta)$

$T_{ij} = E(R_{ij}) = E(Beta(\alpha_{j+1}, \beta_{j+1}))$

$= \frac{\alpha_j + 1}{\alpha_j + \beta_j + 2}$

Update $\alpha_j^{new} = \alpha_j + 2$ coop. (w * α_j) $\quad \beta_j^{new} = \beta_j + 1$ uncoop

$D_{ij}(r = \_, s = \_)$ triple

Aging

2nd hand info

$$\begin{matrix}\alpha_j^{new} \\ \beta_j^{new}\end{matrix} = \begin{matrix}\alpha_j \\ \beta_j\end{matrix} + \frac{2\alpha_k}{(\beta_k + 2)(\alpha_j^k + \beta_j^k + 2) + 2\alpha_k}$$

reputation of k

weight

B_j^k $\quad \alpha_j^k$

! independent Rep Infon.

e.g. $(R_{ij})_D$

propagation

cooperative

TH

indirect observation through node k about j

$(R_{ij})_D \rightarrow RTD_i$ $\quad RTD_i^c$ $\quad RTD_i^{nc}$ no bad-mouthing

refresh (=0) after broadcast

only propagate nodes in RTD_i^c and RT_i^c

OR NOR → forwarding notes 13

Simulation (1,0) (0,1)

aging = 0,98

TH_{STH} = 0,9

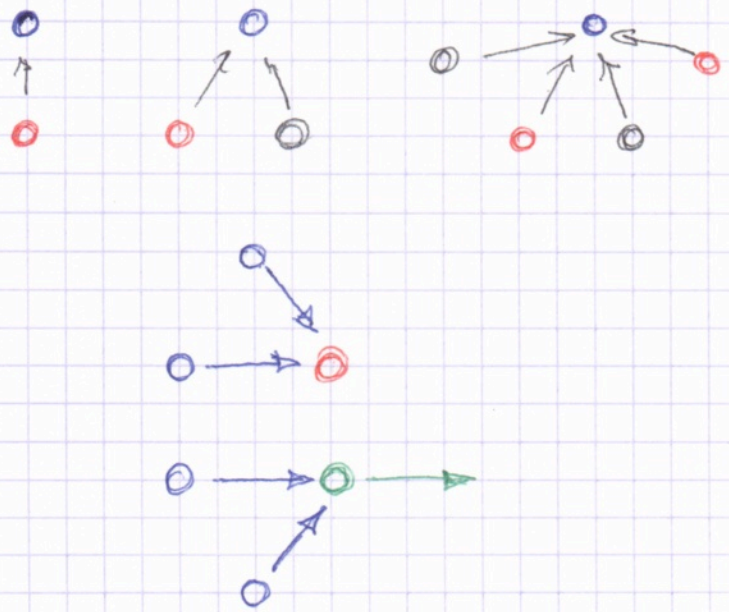
Needs infos on packets send by other nodes

→ on ⟨packet⟩ ⟨trigger-cb⟩ (+ promiscuous mode)

data storage for other nodes

→ variables, dynamic ~ # neighbour nodes
↳ max?

math functions



PoC 2 Cooperative IDS (paper 2)

Model → proofs for IDP + implementation for real.
↳ if impossible in theory ⇒ not realistic. possible in reality = weaker

$S = \{ s_1, s_2, \dots s_n \}$ Sensors.

$N(s)$ Neighbours ⇐ static + symm.
↳ 2-hop.

t nodes attacked ⇒ Byzantine failure.
↳ Byz. Agreement protocol.

$t=1$ → $t > 1$ is hard :->

source(s) ⟺ s = attacker / captured node

honest(s) ⟺ ¬ source(s)

Alert Module $D(s)$ → alerted node → alerted set
suspected | sensors set

$|D(s)| = 1$ ⇒ attacker identified

honest node $\exists s' \in D(s)$: source(s')

s' = attack → δ delay $\exists s : D(s)$

$\forall s \in S$: honest ⇒ $D(s) \subseteq N(s)$

iDP $\forall s, s' \in S$: honest$(s) \wedge$ expose$_s(s')$

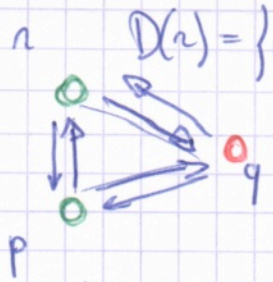
Correctness ⇒ $A(s) \wedge$ source(s')

Termination attack ⇒ after T all honest nodes have attacks in exposed.

$D(q) = \{p\}$ "claim"

$D(p) = \{q\}$ → knows it's honest ⇒ discard

⇒ expose$_p(q)$



$D(r) = \{p, q\}$

$D(q) = \{p, r\}$

$D(p) = \{r, q\}$ ⟹ not solvable → what is { necessary / sufficient

$$AN(s) = \{ t \mid A(t) \land t \in N(s) \}$$

$$\tilde{AN}(p,q) = AN(p) \setminus \{q\} \quad (= \text{valuable to } q)$$

ibC

$$\forall p,q \in S: \ source(q) \Rightarrow \tilde{AN}(p,q) \neq \tilde{AN}(q,p)$$

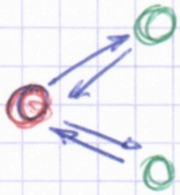
"no other node has same alerted neighbours as attacker"

(proof) ↓ (not necessary)

NC

NC_1 all neighbours of attacker are alerted

and NC_2 2 or more suspects by majority
⟹ honest suspected nodes have non-alerted neighbours



Byzantine (cite)

$n \geq 3t + 1$

$t = 1$ ($n \geq 4$)

Algorithm

1) honest / alerted neighbours of attacker share views
2) agree on source id
3) export it

1/ init phase

a) preload one way key chain length ℓ (before deploy)

$(K_0 \ldots K_{\ell-1} \ K_\ell)$ unique per node — sha 1, sha 1, sha 1

b) discover all neighbours.

c) 2-hop neighbourhood TTL = 2 ↳ table

d) announce K_0 to 1 hop, 2 hop

2/ voting phase (+ timer)

$m_v(s) = D(s)$

$MAC_{K_j}(m_v(s))$

notes 16

3/ publish key phase

previous committed key

$k_{j-1} = sha1(k_j)$

1/ check is next
2/ check authenticity votes

4/ exposing the attacker phase

5/ external ring reinforcement phase

iDC fails

honest nodes have majority ⇒ non alerted neighbours

NC holds

$P = \{ p_1, \dots p_k \}$

non-alerted neighbours of alerted node
↳ request by alerted region with P
↳ reply with "favor" for $\in P$

NEEDS [storage keys (→ flash)]
storage key | neighbour

send packet with π
respond to ↑

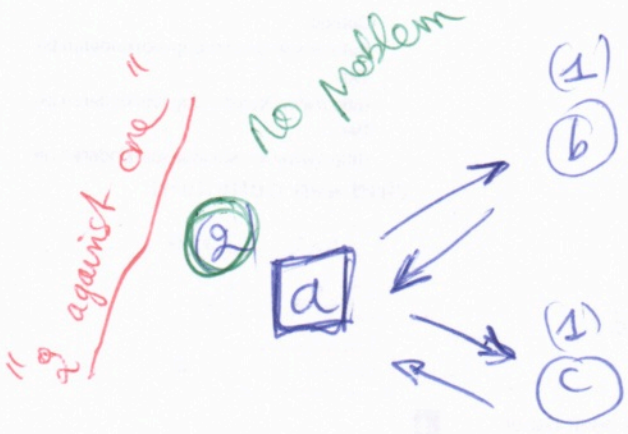
iDC calculation

timer callback

SHA1 function

Notes 17

IDC



$\tilde{AN}(a,b) = \{c\} = \tilde{AN}(b,a) = \{c\}$

$\tilde{AN}(c,b) = \{a\} = \tilde{AN}(c,b) = \{a\}$

$\tilde{AN}(a,c) = \{b\} = \tilde{AN}(c,a) = \{b\}$

$\tilde{AN}(b,c) = \{a\} = \tilde{AN}(c,b) = \{a\}$

$D(a) = AN(a) = \{b,c\}$

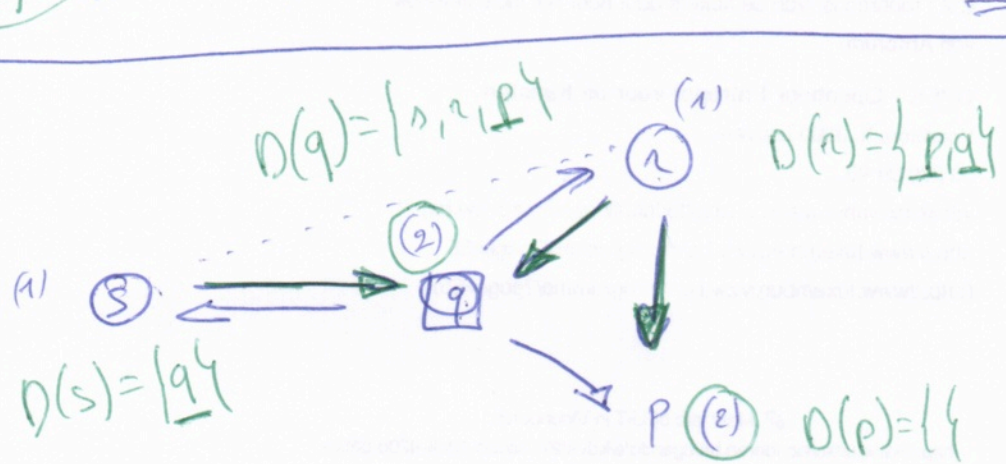
$D(b) = \{a\}$ $AN(b) = \{a,c\}$

$D(c) = \{a\}$ $AN(c) = \{a,b\}$

NC

NC_1 ok

NC_2 holds want geen 2 gmeen $\Rightarrow$



$AN(s) = \{q\}$

$AN(q) = \{s,r,$ [illegible]$\}$

$AN(r) = \{q,$ [illegible]$\}$

$AN(p) = \{q,r\}$

$\tilde{AN}(q,r) = \{s,$ [illegible]$\}$

$\tilde{AN}(r,q) = \{$ [illegible]$\}$ $\not\Rightarrow$ ioc holds. $\Rightarrow$ pof r is aanwezig

ok

$\tilde{AN}(p,r) = \{$ [illegible]$\}$

$\tilde{AN}(r,p) = \{q\}$ $\neq$

ok

↳ Stel s en r neighbours

$\Rightarrow$ $AN(s) = \{r,q\}$ $\tilde{AN}(s,q) = \{$ [illegible]$\}$

$AN(r) = \{s,q,$ [illegible]$\}$ $\tilde{AN}(q,s) = \{$ [illegible]$\}$

$AN(q) = \{s,p,$ [illegible]$\}$

"een van de twee is altijd de ioc"

Notes 18

~~NC1 $\forall a_1, a_2 \in S$: source(a) $\wedge$ $a_2 \in N(a_1)$ $\Rightarrow$ $a_2 \in A(a_1)$~~

~~NC2 $\exists S_1, S_2 \in S$ ...~~

$D(s) = \{q\}$

$D(q) = \{s, r, p\}$ not neighbours

$D(r) = \{q, p\}$

Stel s honest

$\Rightarrow$ source $\in D(s)$ } s $\in$ oscillator $= \{q\}$

en $\in D(r)$ }

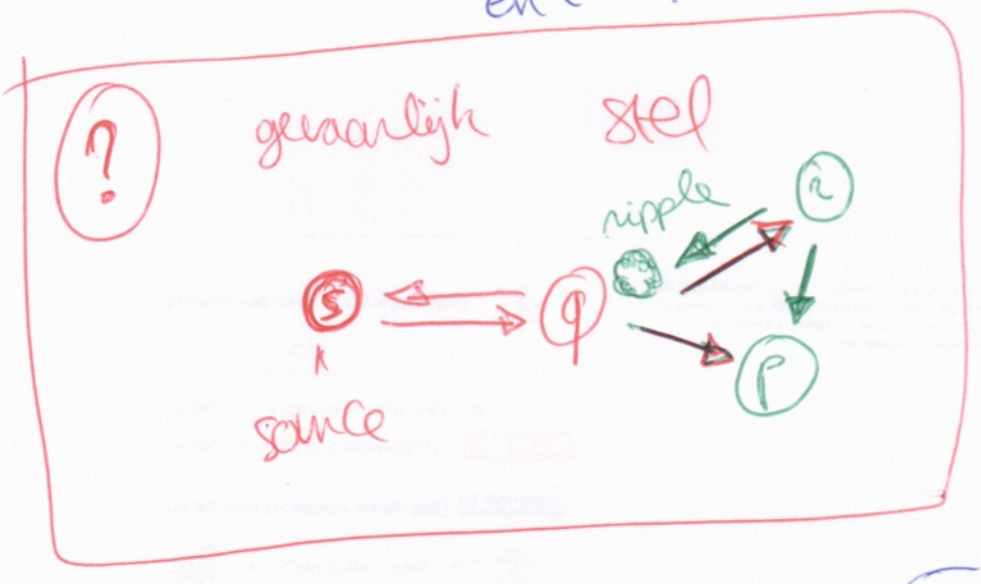
Stel q honest

$\Rightarrow$ s... $\in D(q) = \{p\}$

intersection.

en $t = 1$.

q en r neighbours.



$AN(s) = \{q\}$

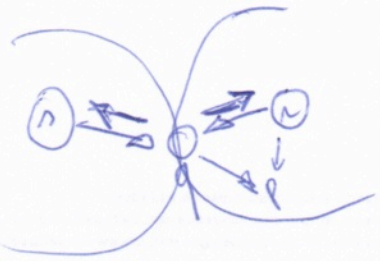
$AN(r) = \{q\}$

Stel s = attack

$s \notin D(r)$

$\{q \in D(s)$

$\{q \in D(r)$



Software Attestation

① Shortcomings.

base station performs SA

* tamper-resistant HW — COST!! — solution? — presumed safe.
* challenge-response

① Rootkit based on ROP Hash

② code compression.

Adversary + situation

1) full control
2) during SA: not full
3) injection ← "buffer overflow" + JTAG ← software vulnerability
4) no network interaction
5) X HW change
6) verifier knows HW + config.

challenge/response

H(·) + nonce

pseudo-random → selection of mem words.
↳ seed from verifier

notes 20

SWATT → timing → delays
software Attestation redirection
limited to program memory
not data / external.

rootkits

Fill empty program memory
↳ compression
only program memory

don't use

Self-modifying code based att.
↳ difficult.

ICE → SAKE
indisputable code execution
↳ program counter needed → ✗ AVR
attestation on attestation
modified on different mem. location.

Rootkit - modify attestation code
- JUMP to hide function
- ROP to "replace"

TinyOS
not Turing complete needed
→ bootloader

Compression Attack → room for.

memory shadowing

compress + on the fly decompression
program mem.

Time-based ↔ SWATT

↳ avoid using redirection = introduces delay (can be noticed)

e.g. attestation 23 cycles
+ redirection 3 cycles = 13%

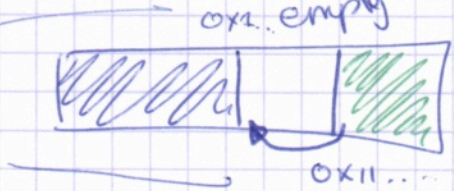
1) empty mem redirecting using bitflip

0xFF → 0x1xxx...x

0x11xxx...x

= 2 instructions

= 7,4% = 43% faster than

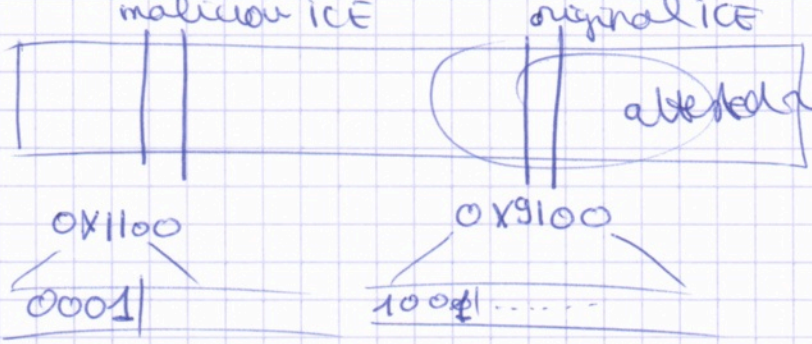


Data memory attestation

→ data memory is unpredictable

ICE

displacement ⇒ MSB of PC.



ICE

T function → random permutations of mem. locations.

mem loc → 160 bit checksum C
of 10 16bit registers. C_j

$$C = [C_0, \dots C_9]$$

program counter.

$$C_j = C_{j-1} + PC \oplus mem[current\_address]$$

$$+ j \oplus C_{j-1} + x \oplus current\ address$$

$$+ C_{j-2} \oplus SR$$

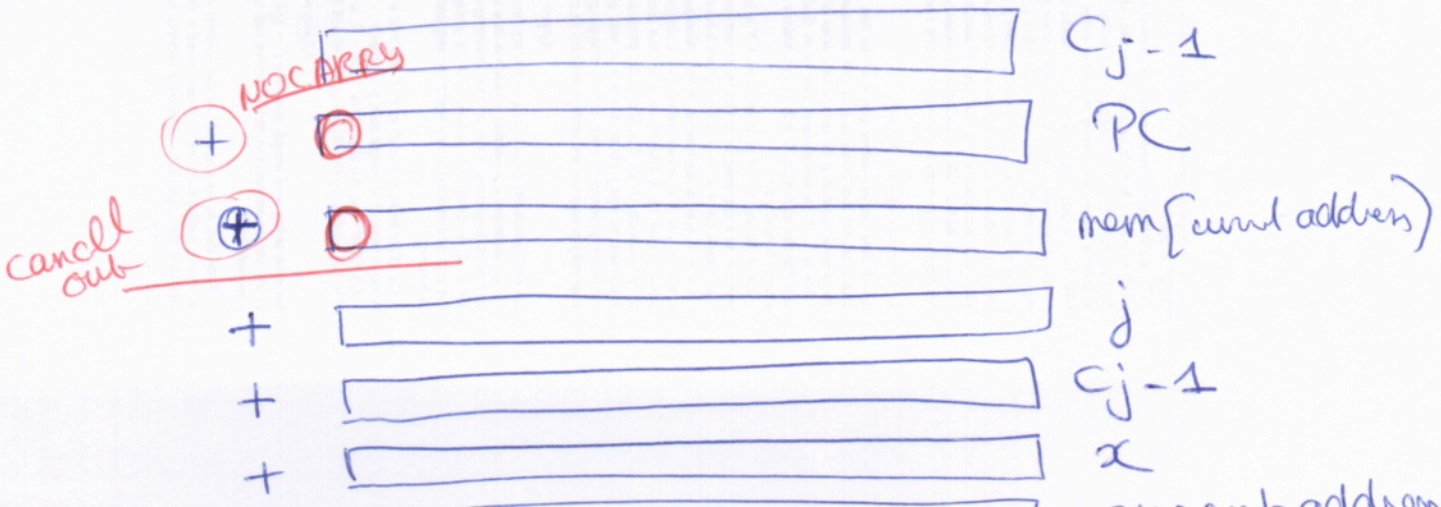
loop counter

last value from T function

status register.

$\oplus$ = 16 bit XOR

$+$ = 16 bit SOM (no carry)

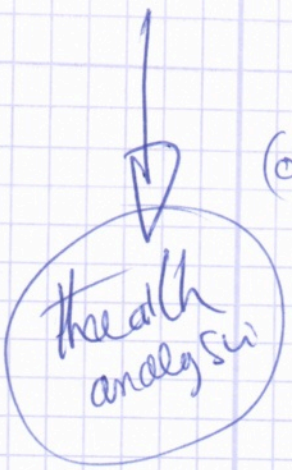


notes 23

Categories in/out-sider attack
mote/laptop class

blablab

aschenbruch



Goals (of attack)
- manipulate data — Integrity
- eavesdrop — Confidentiality
  ↳ ROM/RAM dump
- drop data — Availability
- network access.

Methods - rushing - DOS

Situation
- resource constraint (energy)
- fysical access (node capture)
- in-network process (no end2end encryption)

Valeros

Attacks
* jamming
* Sinkhole → false ETX count → attract data and drop
* wormhole → faste redundant connection cross network = replay attack
* Sybil
* selective forwarding

prepare

Goals
- eaves dropping
- traffic Analysis
- Disruption
- Hijacking
- Physical attacks

notes 24

WSN Properties

(blilat)

1)
2) Aggregation → Tree structured routing
3) → Tolerable failure
4) → In network processing
5) → Sensa = router
6) → Phased transmission periods.

Security goals auth., access control, confidentiality, integrity, authentication, anonimity, non-repudiation freshness, availability; resilience

Security in WSN

perrig

High level

1) Secure group mgmt
2) iDS ← in case of failure
3) secure data aggregation

fundamental issues

1) node capture
2) physical tampering
3) DOS

notes 15

IDS = "capturing audit data and reasoning about the evidence in the data to determine whether the system is under attack".

Zhang 2000 intrusion

system → scope? → NODE? NETWORK?

Network ⟷ Host

Misuse ⟷ anomaly

misuse → patterns to detect

anomaly → abberations from detected pattern (expected values)

difference?

e.g. 'moved node: wrong routing info' or captured?

→ Agents local, distributed decision.

Anomaly Detection

Cooperative detection → state info exchange + majority rules

→ definition in conclusion

IDS arch should be
- distributed / cooperative
- statistical anomaly
- cross layer NW

aschenbuch

heartbeat → node removal
↳ interrupted working

↓

anomaly → requires "control"
↳ distributed all { nodes / neighbours

movement → HW : e.g. { accelerometer / PIR

↓

(anomaly)
event

CST ~~at~~ → Carrier Sense Time

→ jamming detection.

↓

anomaly threshold

$t_n = (1-\alpha)\, t_0 + \alpha\, s$ (aging)

alarm if $s > t_0 + w$ (dd, sample, std dev. over all samples)

OTAP : { reprogram / protocol + ECC TinyECC

padmavathi 2005

notes 27

Passive → =information gathering: monitor / eavesdrop
Active → =change

Traffic analysis / sniffing, jamming

no physical
physical access = node capture / move / remove

reprogram node ← OTAP or → Soft-vulnerability

access node memory ← "buffer overflow + code to send"

Normal physical access e.g. RS232

JTAG, ...

inject node
physical / logical

capture network / key info

network access

introduce hardware

sybil = not in itself, but base for actual attacks
= ability to create nodes

wormhole → no actual participation, just mirror

sinkhole → participate

a) laptop
b)
c)
d)

flushing → attract traffic
a) ignore delays / timeouts
b) fake routed packets

denial-of-sleep

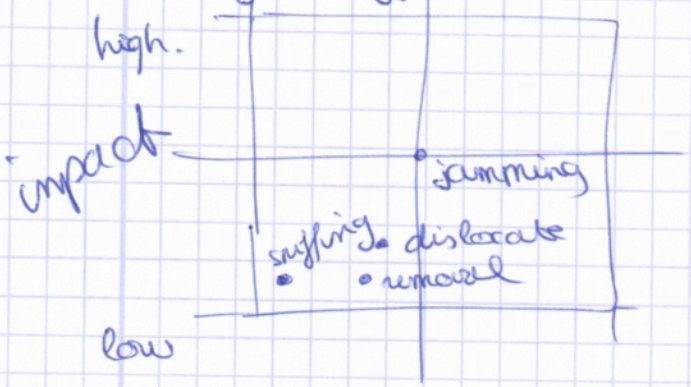
replication → base for e.g. sybil

① goals
- ① collection of data / eavesdropping / stealing
- ② alter function = render WSN useless
- ③ DOS ⇄ openlijk / covert (denial of sleep) — results.

② strategy
- attract traffic ≤ sinkhole, wormhole, sybil — verkeken
- analyse "
- "jamming" / verstoren ← {drop message, selective fwd, node removal
- reprogram node

doel op zich — ondersteuning

③ methode ways
④ techniek
?! sympathieke way

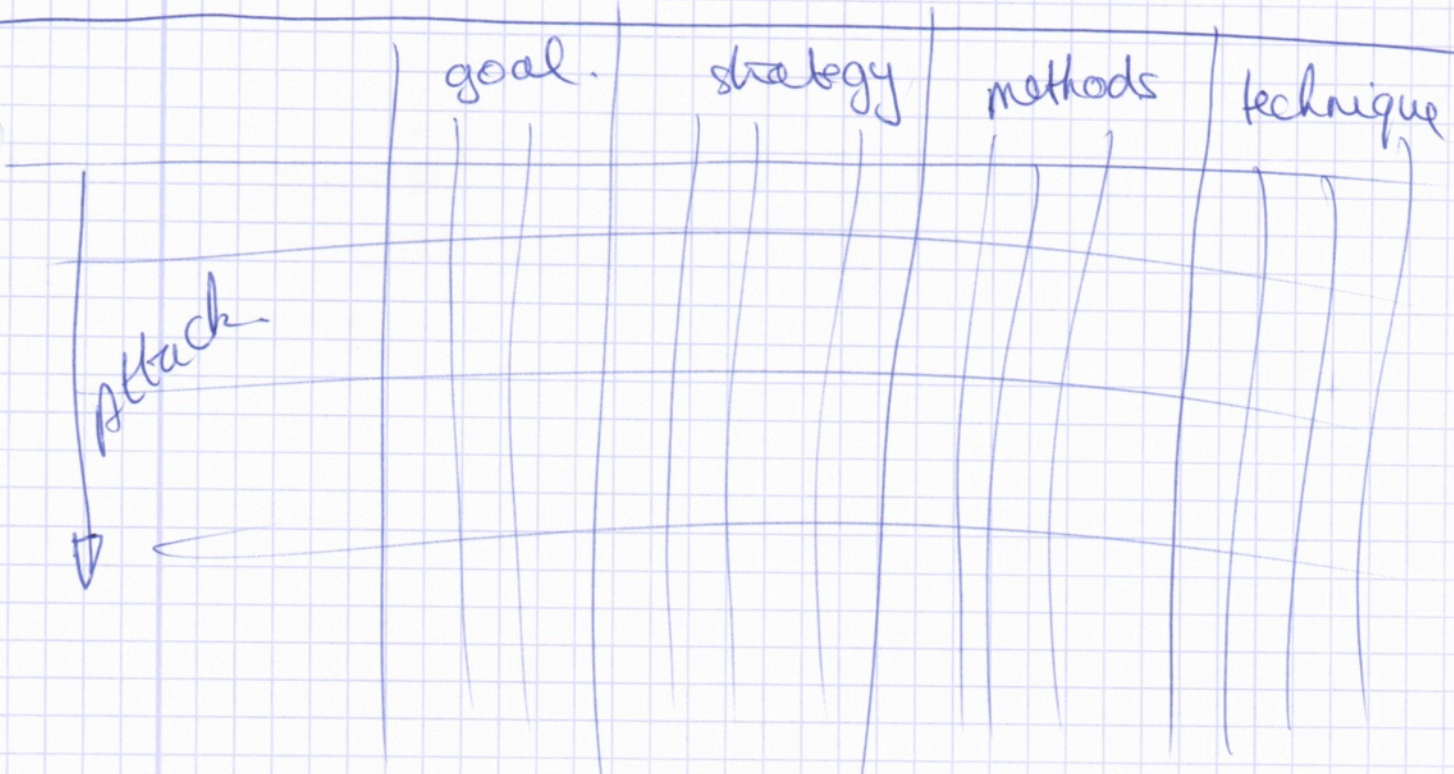


idee

+ dependencies

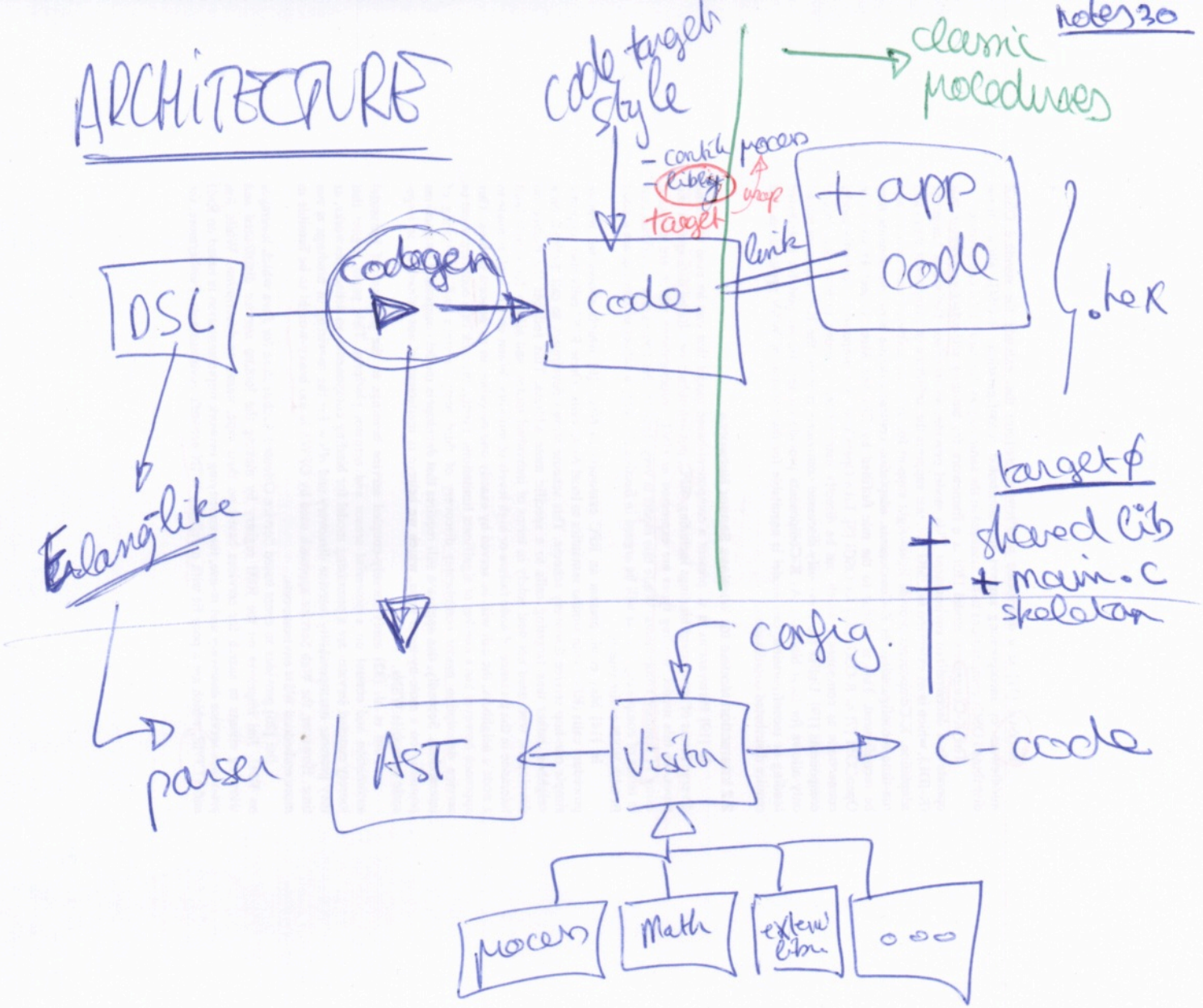
requires

enables



ARCHITECTURE

notes 30



ATTACKS

corruption

- Sinkhole
- Wormhole
- Sybil
- selective forwarding
  - ↳ cooperative ("anomaly")
- hello flood

~~snuffing~~

jamming → anomaly

rushing → anomaly

Denial-of-Sleep → anomaly

↳ move accelerometer (HW)

Node
- dislocate
- remove
- insert
- replication

→ anomaly (e.g. heartbeat)

~~access memory~~

reprogram ↝ "anomaly" /(SA)

---

cooperative decisions → generic FW?

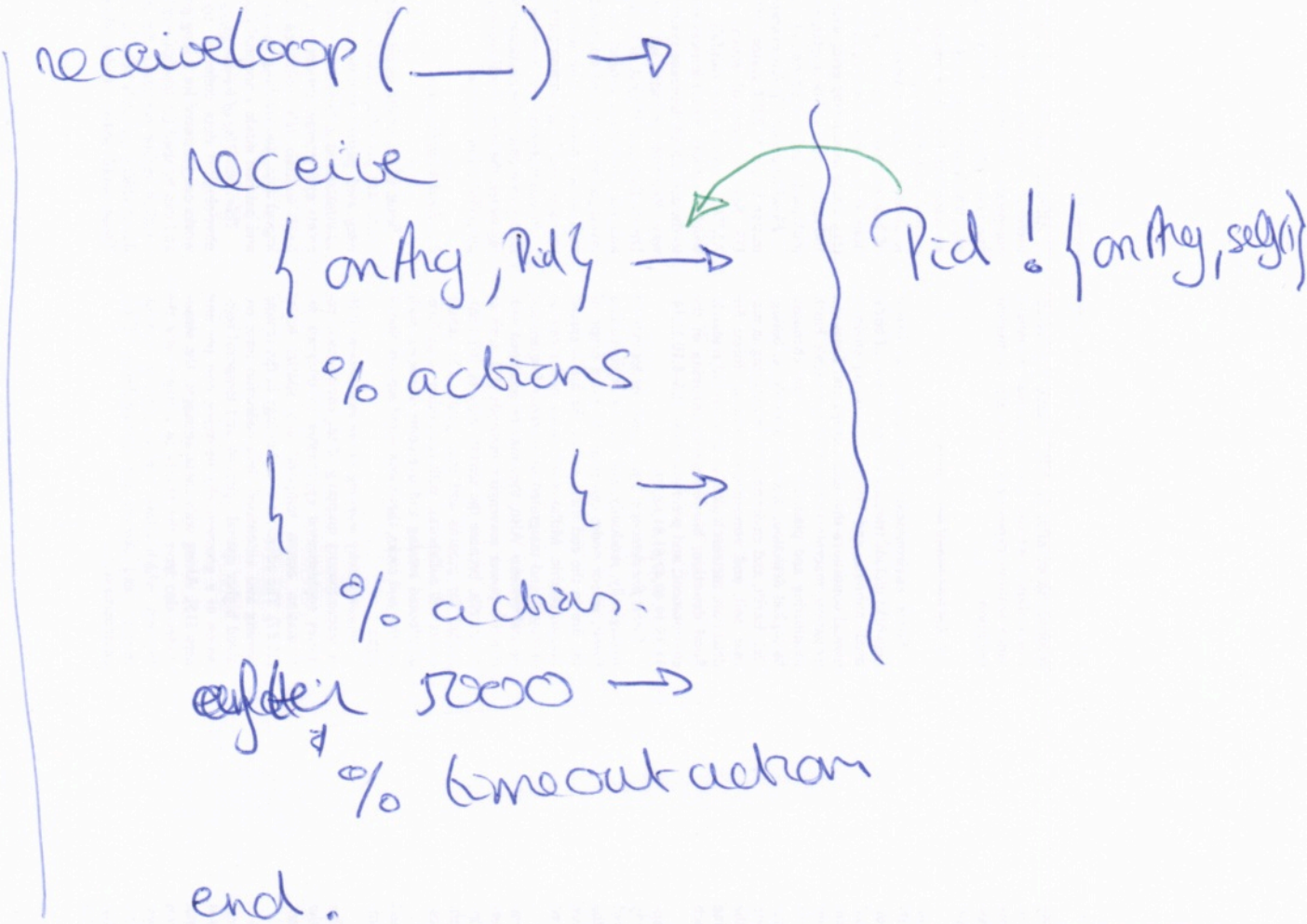
"HIDS" sanity checks (combo ICE-style)

---

malfunction!?

ERLANG (-style) DSL

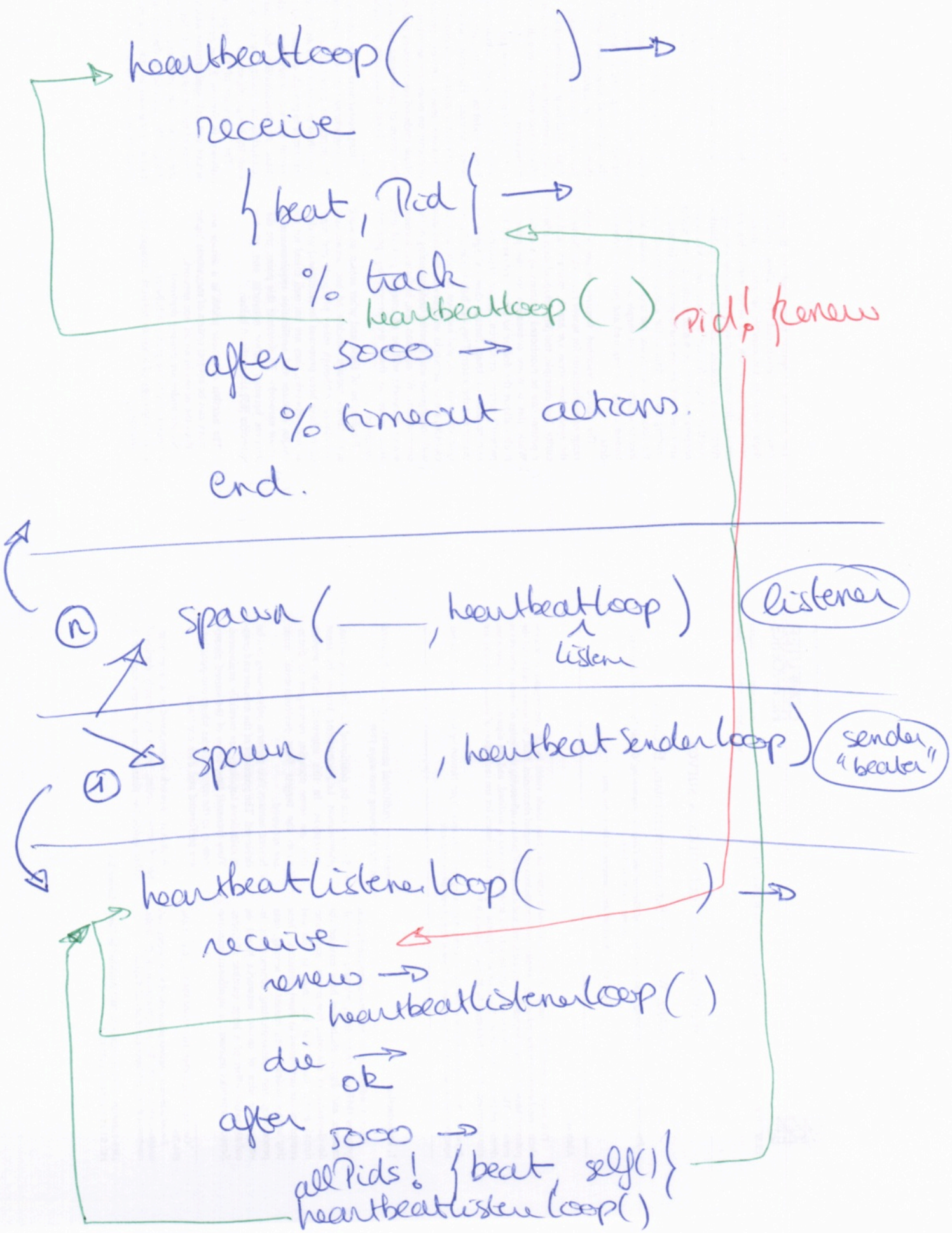
communication



spawn

spawn(module, function

example heartbeat



example heartbeat "pseudo code"

Strategy event driven

↳ event table

process events → "near" realtime (very near :-))

event "wait for heartbeat from Pid, timeout, epoch

event "heartbeat from Pid, time, epoch

↳ serial synchronisation

↳ process stack-style → pop, merge, ...

↳ VM

nodes 35

modules allow for expansion

Use `nodes` — "variable" with module of funcs

type?

`nodes.nodetype = { key : type = value }`

build-in hash type

also [] exists.

app specific node info

extends "basenode"

↳ includes – unique id

`nodes.nodetype.handler = <fun>(node)` ?

function to "handle/process" nodes

`.visitor`

`.handle/condition` — is valid? principle of event

`.internal` — how deal with other nodes? lookup → no loops

combo scheduling?! validity? too restrictive

data receive event

hash (impl. struct)

```
nodes.on_receive(from, payload) {
    nodes[from].property = payload.property
}
```

basic logic available

math: +, -, *, /, sin, cos, ... } reuses C

bool: and, or, not

cond.: if then else

switch case.

---

other modules: "possible"

- sensors
- actuators
- crypto (e.g sha1, ...)
- memory → e.g for SA
- time (e.g. time.now())

"functional"

<node> ← { payload }

broadcast ← { payload }

pub/sub?

send to operator

receive spec

"event handler"

node → { "payload pattern match" }

→ (?) "event raising" raise( )

"ch exception handling" :-)

e.g. <do . . . >

after 50000

raise ("not ok")

what's different for function call?

→ handle 'not ok' { . . . }

---

reading not repeatable

"synchronous send style"

send_something (node);

receive

{ type = monitoring, from = , payload = }

→ ok

after 20000:

("whoops did not send") ⇒ nodes.forwardfails+

Time of point "after new start" ⇒ . . .

explicit even an asynchronity

✗ ✗ → ✗ → ?

send (node, payload), after (send) = {

receive . . .

after . . . }

∅ omittable

DSL container → DSL to {implement} {structure event-driven and code-accepting DSL's} targetting ↳ code generation

const

use "DSL"

extend "DSL" with {K : T = V} → typed hash

when "DSL" scope <function> do <function>

<name>
function(...) { ... }

with "DSL" scope do <function>

@<annotation>

every (time interval)

---

language constructs

if (<bool-exp>) {<stmts>} else {<stmts>}

memberaccess <object>.<member>

method call <object>.<method>(<args>)

assignment (<var>) = <value>

+ implicit declaration on first use ?!
+ type inference ?!

math + - * / % ( )

push

lang.

case <variable> contains

list → methods.

notes 38

? why not simply reuse if( ) ...

```
if(<car>.contains) { ... }
elseif(car.contain) {  }
else {  }
```

syntactic sugar ?

TODO ? incomplete boolean expressions.

↳ matching [⊙ < now() , —]

? indicating

don't care "—" "—"

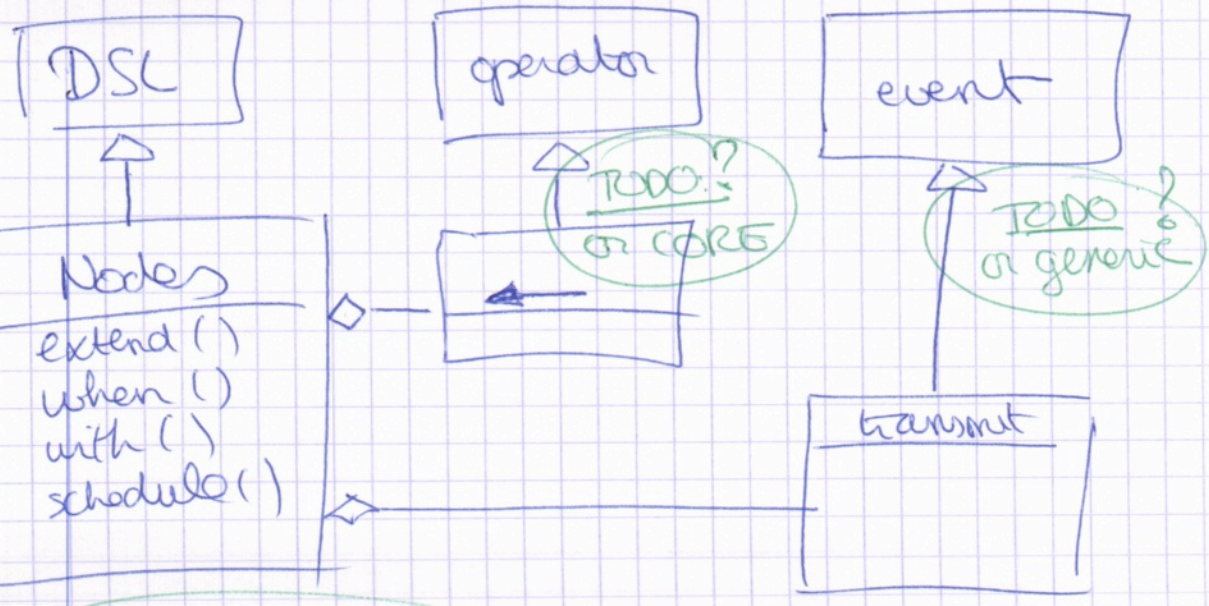
≃ . == .

idem ≙ any.

function(val) { return true }

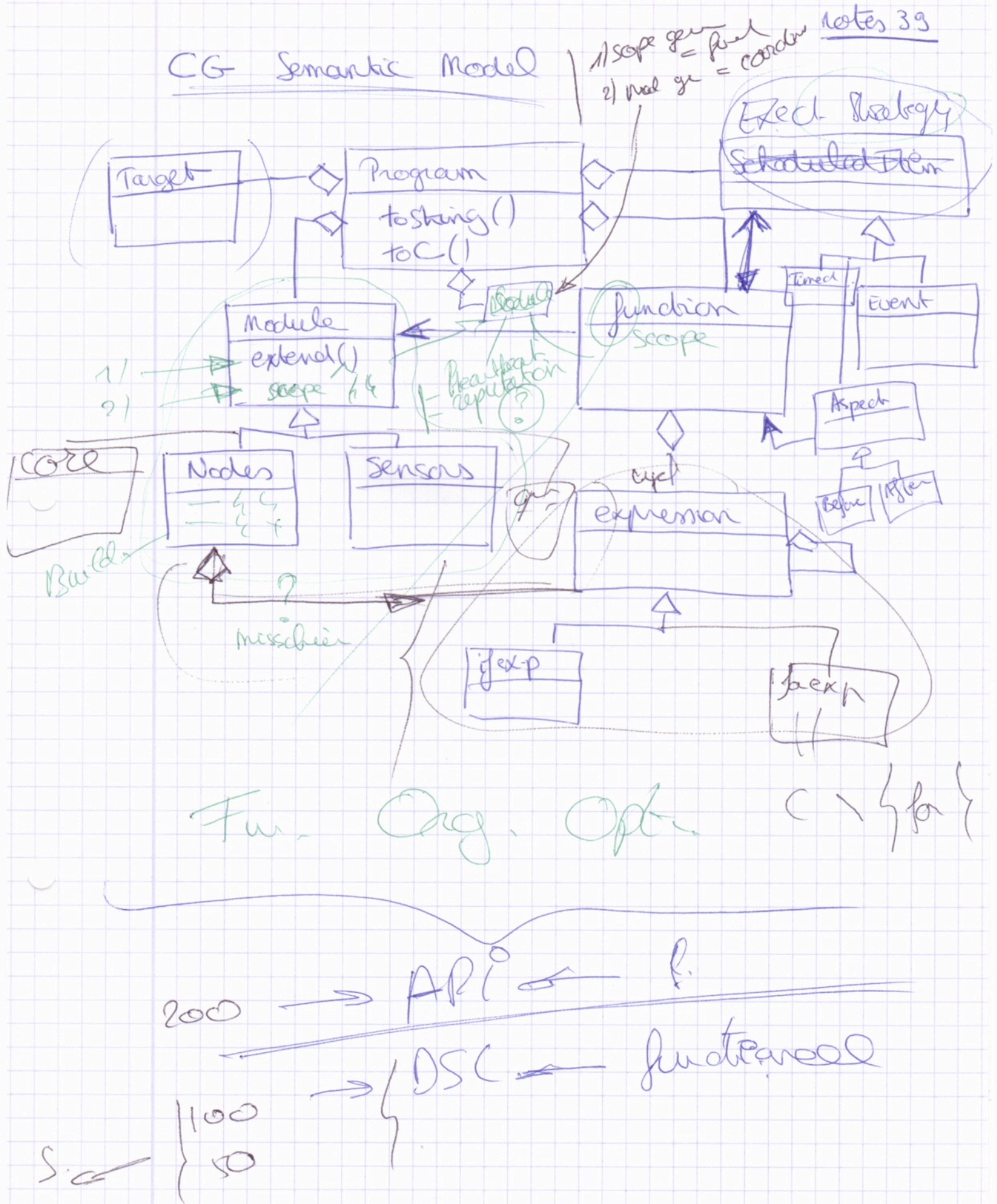
syntactic sugar for closure ?

function(value) { value < now()



CG Semantic Model

notes 39



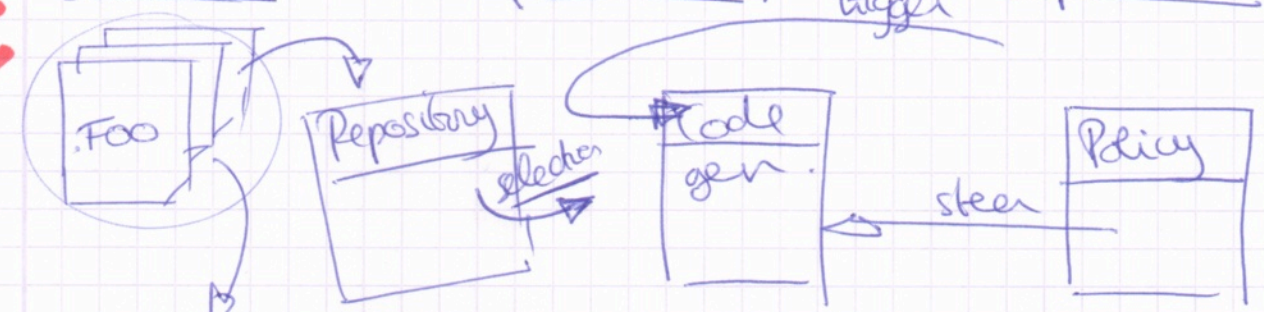
when → after / before ?
↳ event ?
≈ after.
↳ aspect oriented.

notes 40

ARCHi → diagrams for report

FUNC

Research (Develop) Operations

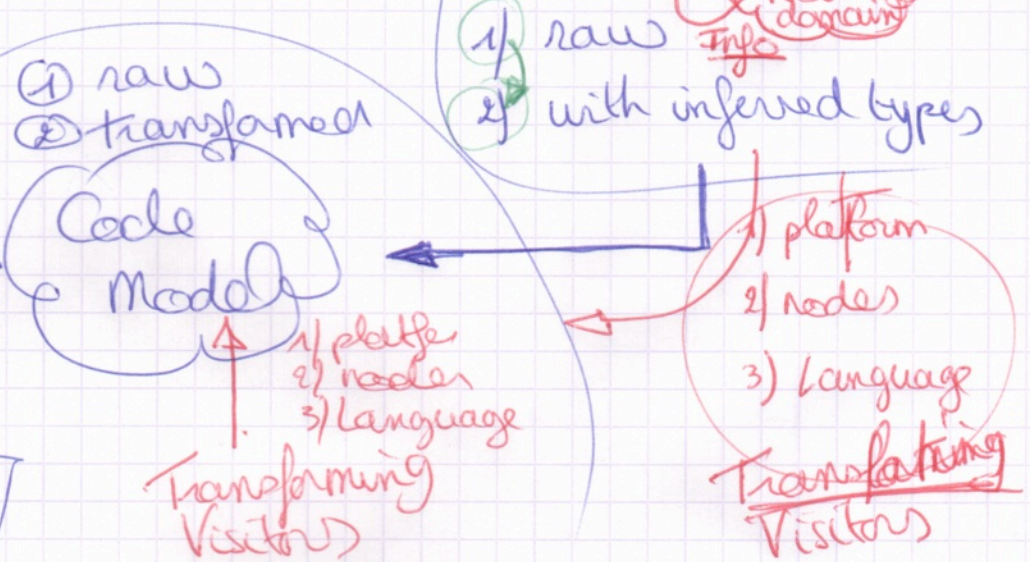


Simulator
interpreter, debugger, ...

+ manual application

Solution → deployment

TECH



→ compilation