KogSys Reading Club SS 2010

Grundlagen der Kommunikation unter Robotern

Martin Sticht

Sinn und Zweck

Warum Kommunikation von Agenten?

Kommunikation von Robotern/Agenten unterstützen das Lösen gemeinsamer Probleme

- Task Sharing
  - Ein Problem wird in Sub-Problems zerlegt und auf Agenten verteilt
  - Die Aufteilung wird kommuniziert
    → Contract Net, Bidding...
- Result Sharing
  - proaktiv oder reaktiv

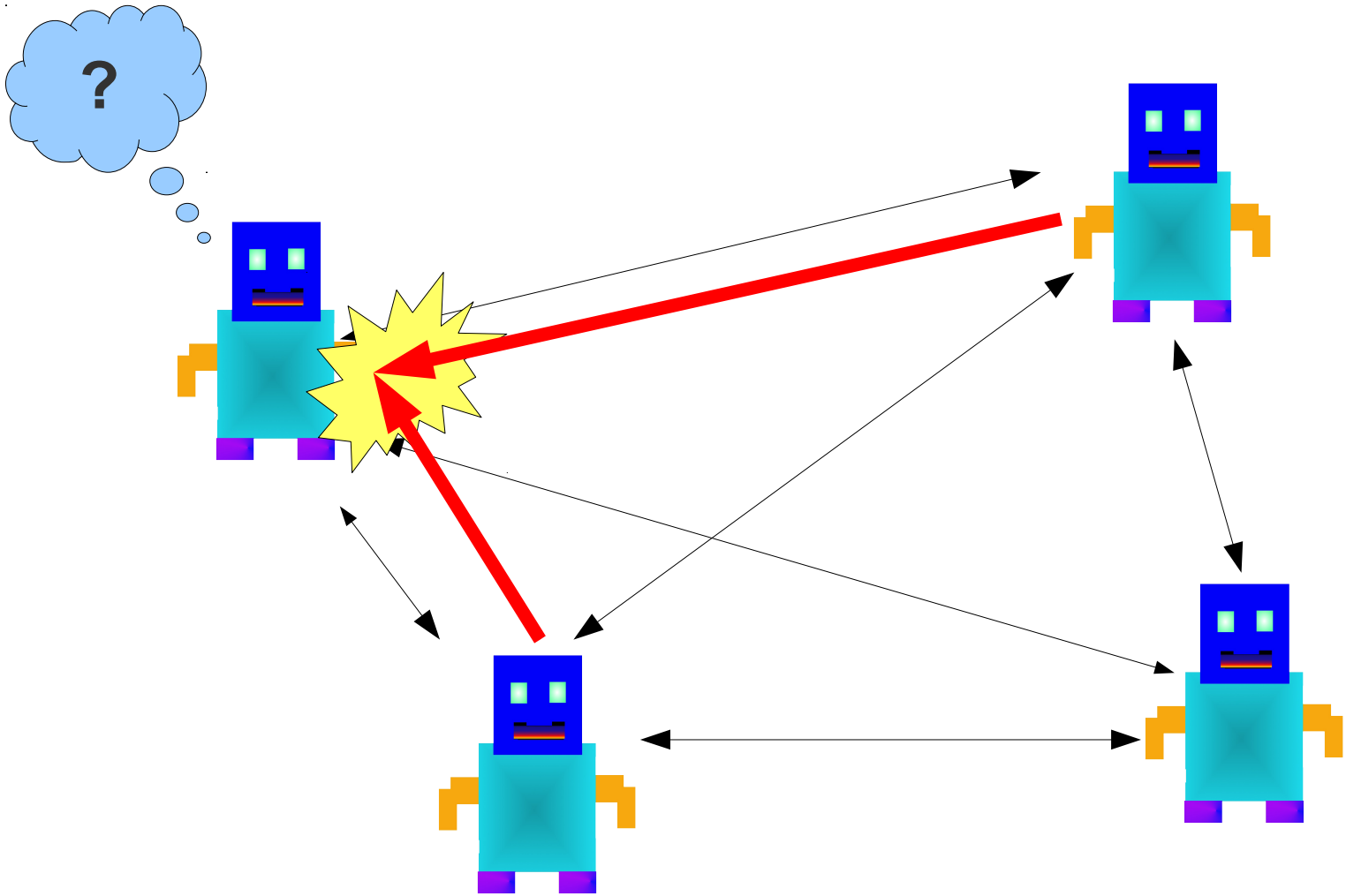
Selbstkonfigurierendes Netzwerk

Arten der Roboter-Kommunikation:

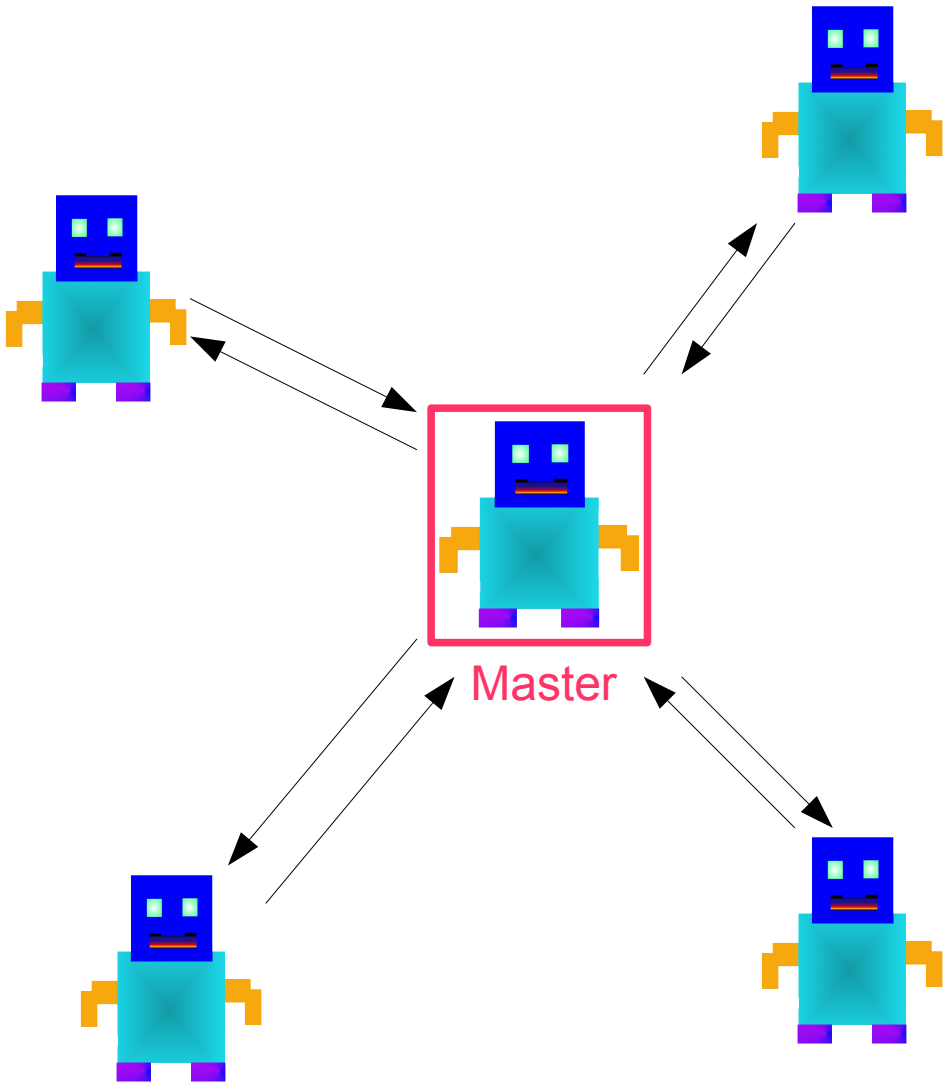
1. Roboter kommunizieren miteinander
2. Roboter werden ferngesteuert
3. Sensordaten des Roboters werden beobachtet
4. Offline-Processing (z.B. Kombination aus 2. und 3.)
5. Monitoring-Console für komplette Welt
   (z.B. Analyse von Interaktionen...)

Im Folgenden:
Kabellose Kommunikation unter Robotern

Kommunikationsmodelle



Polling



Ein Knoten wird zum Master-Knoten bestimmt

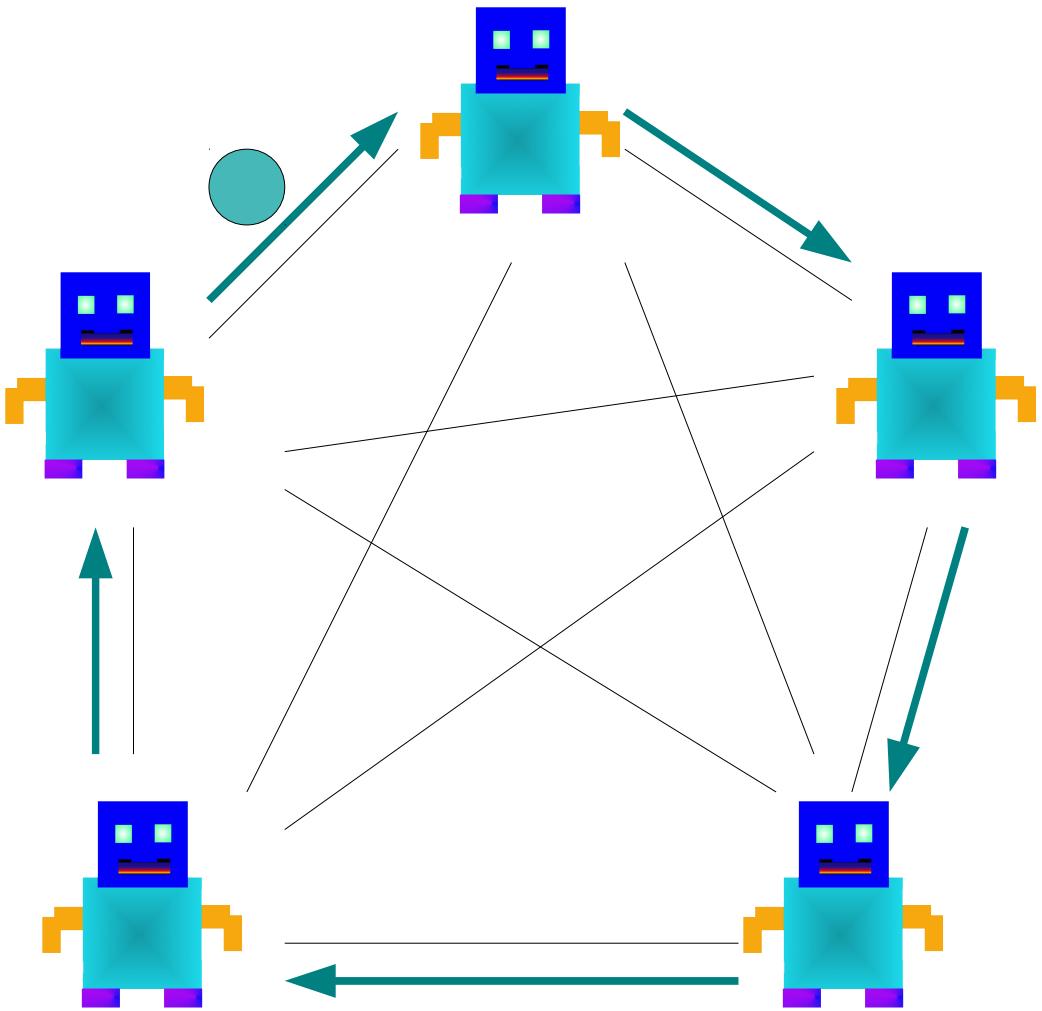
Der Master erteilt jedem Slave der Reihe nach die Erlaubnis, eine Nachricht zu senden

Alle Agenten hören auf alle Nachrichten und ignorieren diejenigen, die sie nicht interessieren

Der Master wird statisch festgelegt.

Was, wenn Master ausfällt?

Virtual Token Ring



Ein Token wird gemäß einer zirkulären Liste von Agent zu Agent geschickt

Wer gerade den Token hat, darf seine Nachrichten senden und muss den Token an den nächsten Agenten weitergeben

Hier kann jeder Agent der Master sein, der das Netzwerk initiiert und bei Problemen einschreitet

Im Folgenden wird der *Virtual Token Ring* als Kommunikationsmodell weiter betrachtet

Aufgaben des Masters

1. Liste aller aktiver Roboter erstellen und pflegen
   Roboter mit Informationen versorgen:
   Jeder Roboter muss wissen, wer sein Nachfolger ist

2. Der Datenverkehr muss überwacht werden:
   Timeout → Token verloren?? → neuen Token erzeugen

3. Roboter mit Fehlfunktion aus Liste streichen

4. Neue Agenten zum Kollektiv hinzufügen und alle anderen informieren

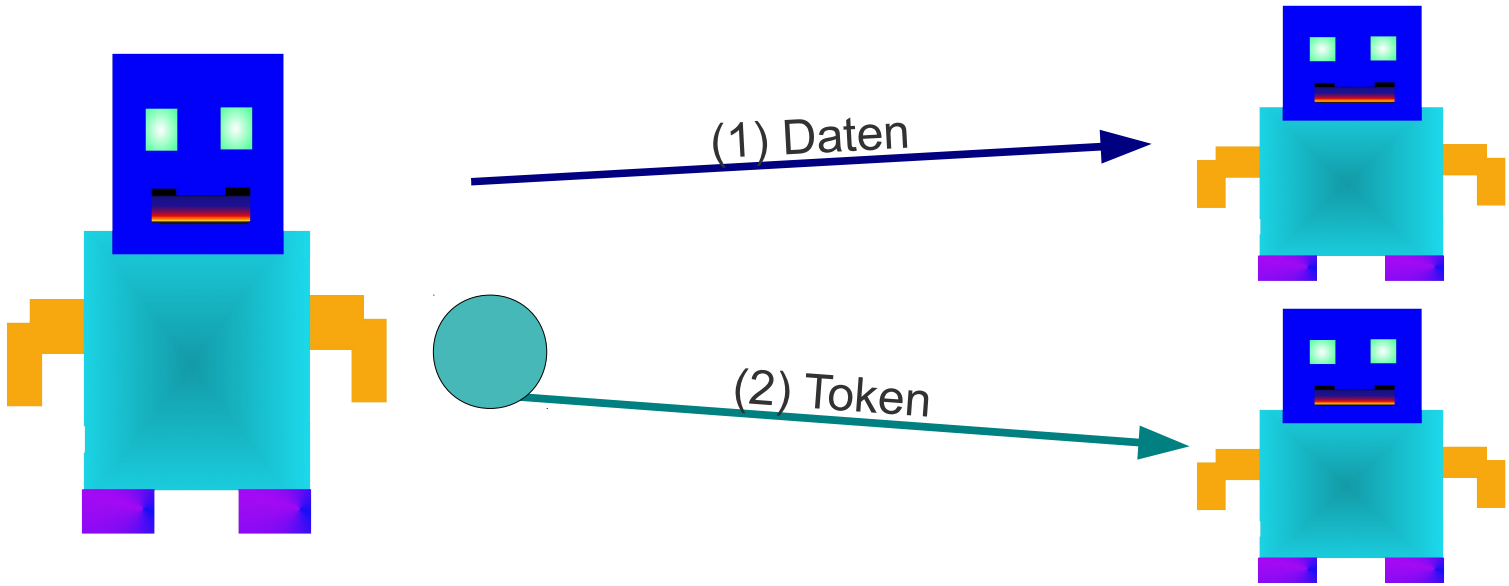
```
Liste:

ID 1 (Master)
ID 2
ID 3
ID 4
```

Ablauf im Token Ring

1. Ein logischer Ring mit allen beteiligten Agenten wird erzeugt. Der Master erzeugt einen einzelnen Token.

2. Der Agent, der den Token hat, sendet eine erlaubte Anzahl an Daten (Frames) und gibt den Token dann weiter.



Mögliche Nachrichten-Frames

Jeder Roboter hat eine feste ID, die sich nie ändert

Inhalt eines Frames:

- **Start byte** — Spezielles Bit-Muster, um Beginn zu signalisieren
- **Message type** — user data, system data, ...
- **Destination ID** — ID des Empfängers; für Broadcast z.B. 0
- **Source ID** — ID der sendenden Agenten
- **Next sender's ID** — ID des nächsten Agenten im Ring
- **Message length** — z.B. Anzahl der Bytes der eigentlichen Nachricht (kann auch 0 sein)
- **Message contents** — eigentlicher Inhalt
- **Checksum** — CRC über gesamtes Frame

Message types

Warum Message Types?

1) Messages exchanged at application program level
   Nachrichten, die zur Problemlösung und Datenaustausch zwischen Agenten verwendet werden

2) Messages exchanged to enable remote control at system level
   (i.e. keypresses, printing to LCD, driving motors)
   Nachrichten werden zur Steuerung verwendet

3) System messages in relation to the wireless network structure that require immediate interpretation

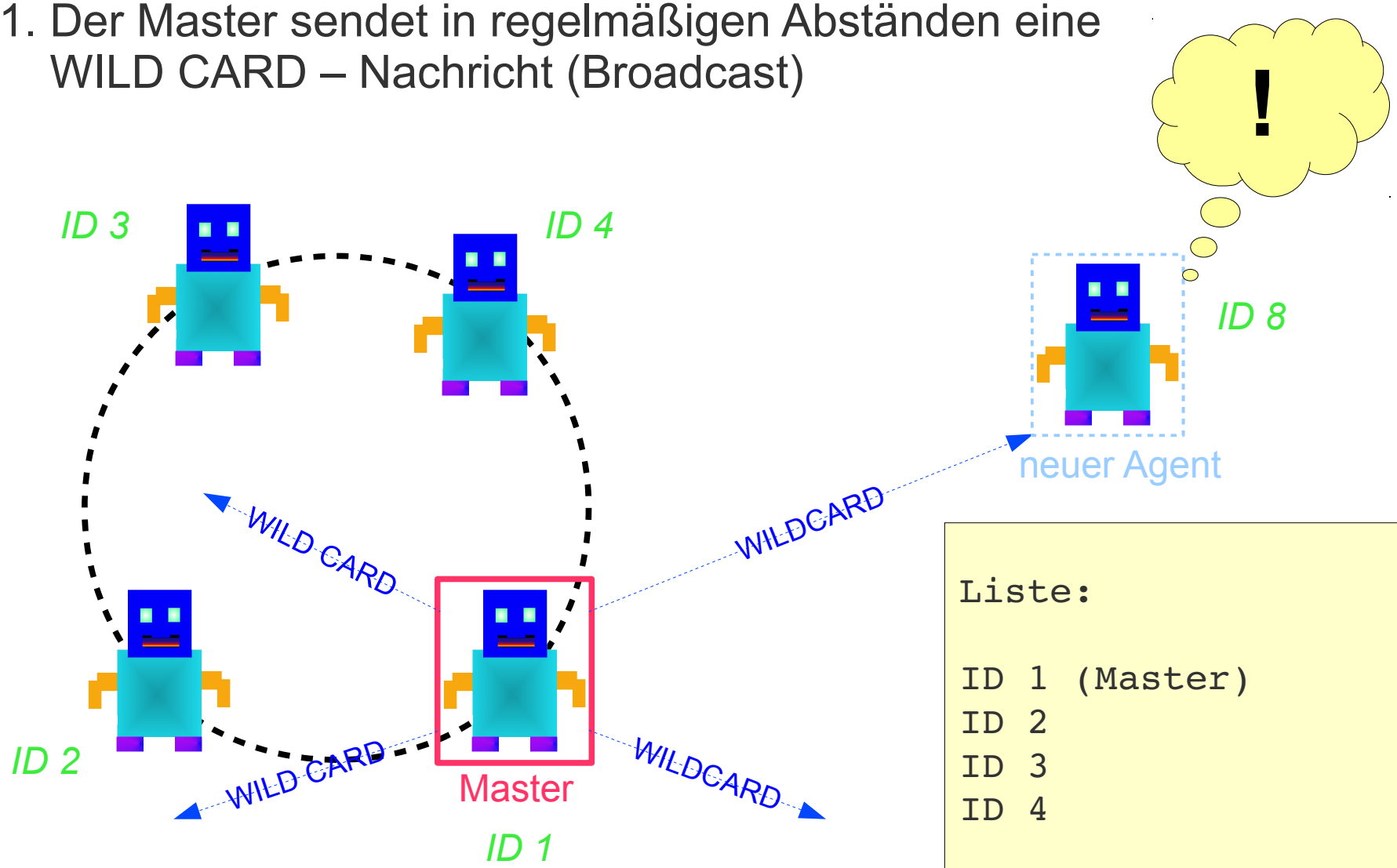
Message types

Mögliche Typen:

USER	Nachricht, die zwischen Agenten geschickt wird
OS	Nachricht vom „Betriebssystem“
TOKEN	ohne Nachrichteninhalt
WILD CARD	wird vom Master gesendet, um neue Agenten zu finden
ADDNEW	Antwort auf die WILD CARD durch neuen Agenten Der neue Agent teilt seine ID mit
SYNC	wird vom Master gesendet, um aktualisierte Liste allen Agenten mitzuteilen

Wie wird ein neuer Agent gefunden?

1. Der Master sendet in regelmäßigen Abständen eine WILD CARD – Nachricht (Broadcast)

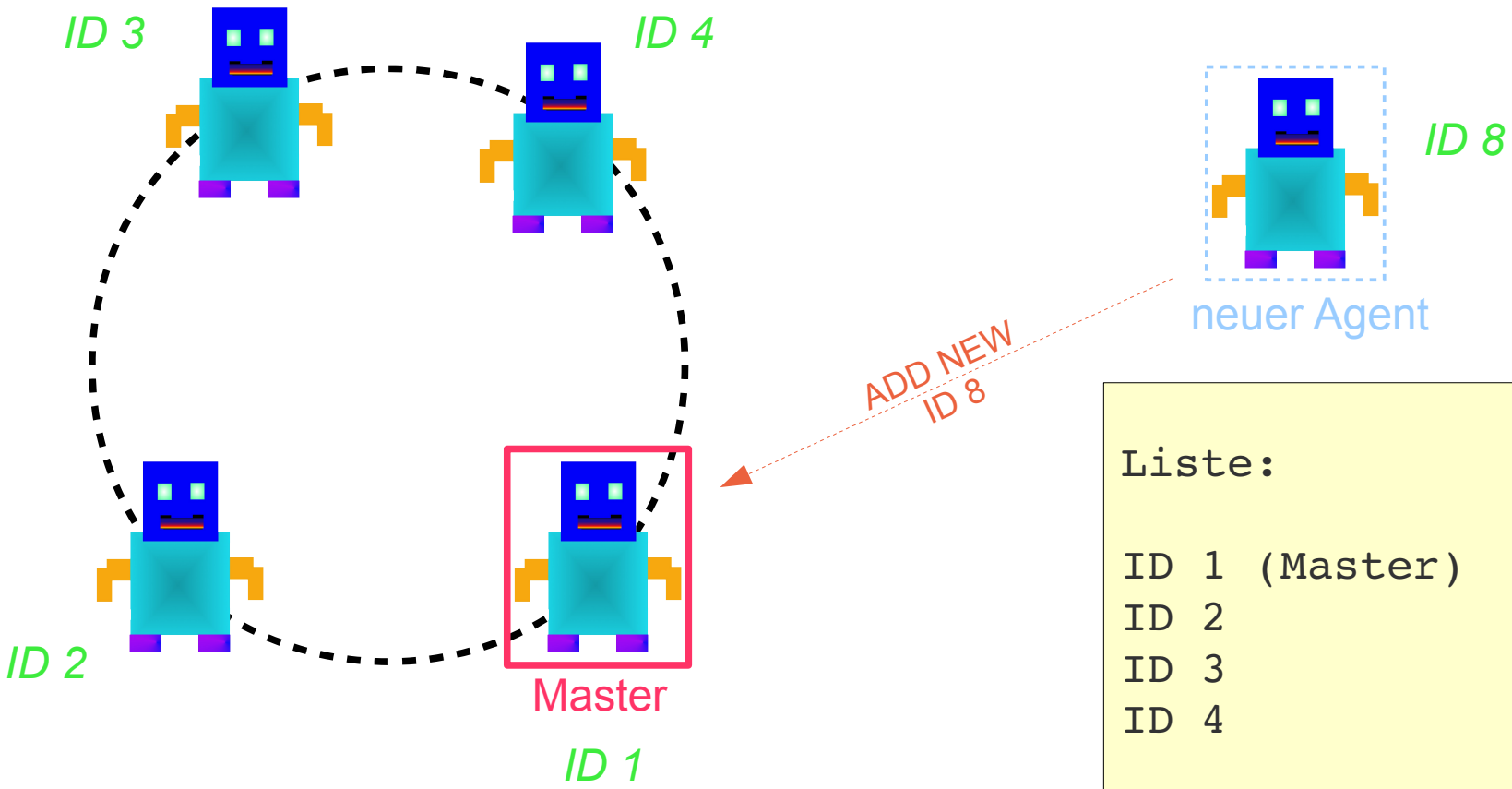



```
Liste:

ID 1 (Master)
ID 2
ID 3
ID 4
```

Wie wird ein neuer Agent gefunden?

2. Der neue Agent antwortet mit ADD NEW und teilt damit seine ID mit

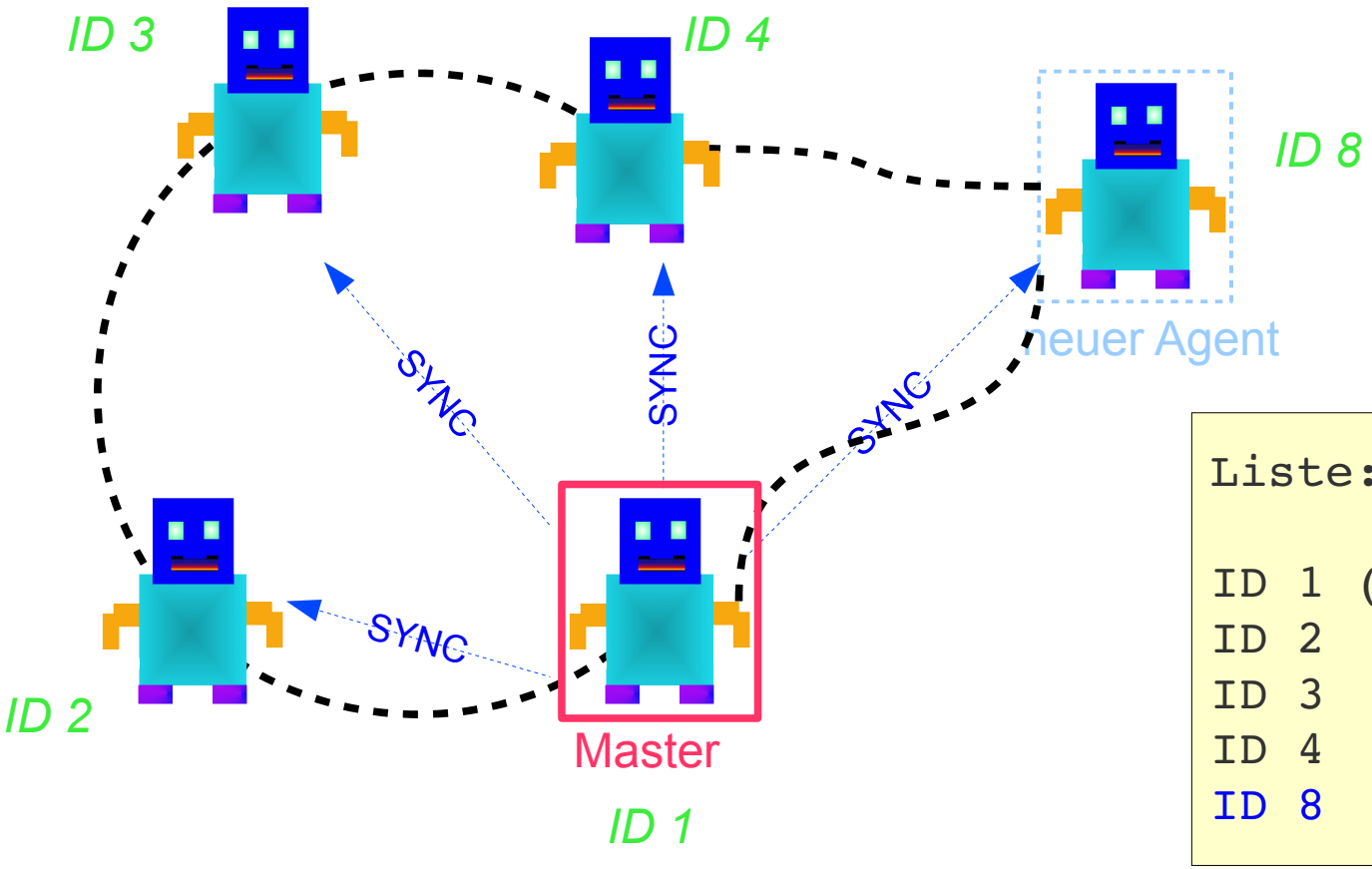



```
Liste:

ID 1 (Master)
ID 2
ID 3
ID 4
```

Wie wird ein neuer Agent gefunden?

3. Der Master aktualisiert die Liste und teilt sie allen mit (Broadcast)

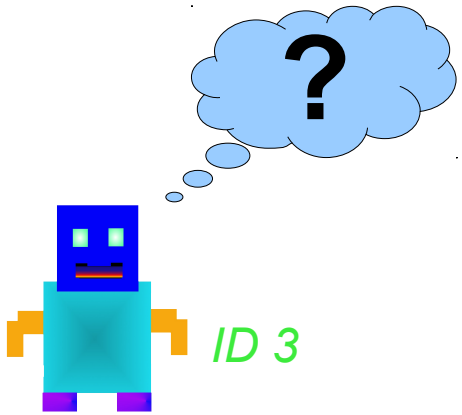



```
Liste:

ID 1 (Master)
ID 2
ID 3
ID 4
ID 8
```

Initialisierung und Selbstkonfiguration

Im Token-Ring kann theoretisch jeder Agent Master sein
Der Master kann auch ein Rechner sein, der alles kontrolliert



- Ein Roboter wird eingeschaltet.....
  - Alles, was er zunächst weiß, ist seine eigene ID
  - Was ist herauszufinden?
    - Wie viele Roboter kommunizieren?
    - Was sind ihre IDs?
    - Wer ist der Master, der den Ring initialisiert und bei Problemen eingreift?
- Der Agent hört eine gewisse Weile, ob irgendwo eine Nachricht gesendet wird
  - Wenn er Nachrichten hört, wartet er auf die WILDCARD
  - Falls innerhalb einer gewissen Zeit nichts passiert, ernennt er sich zum Master
    - Diese Zeit ist das Produkt aus einer festgelegten Zeit und der eigenen ID
      z.B. 10s * 3 = 30s wenn ID des Roboters 3 ist
    - Wenn die Zeit abgelaufen ist, ohne dass eine Kommunikation stattfand, weiß der Agent, dass er die kleinste ID hat
    - Er fängt an, WILD CARDS in regelmäßigen Zeitabständen zu senden
    - neue Agenten antworten mit ADDNEW
      ANNAHME: nur 1 Agent wartet gleichzeitig auf „Aufnahme“ in Liste

Fehlerbehandlung

Jeder Agent kann Zeit messen, um ggf. zu überprüfen, ob der Token noch da ist

- Falls nach einer festgelegten Zeitspanne keine Kommunikation stattfindet, wird angenommen, dass der Token „verloren gegangen“ ist, z.B. durch
  - Kommunikationsfehler
  - Fehlfunktion eines Roboters
  - Ausschalten eines Roboters

- Master prüft ggf., wo der Fehler liegt
  - Master generiert einen neuen Token und überwacht die Kommunikation
  - Wenn der Token an der gleiche Stelle wir zuvor verschwindet, wird davon ausgegangen, dass der betreffende Roboter nicht richtig funktioniert
  - Der „kaputte“ Roboter wird aus der Liste gestrichen

Fehlerbehandlung

Jeder Agent kann Zeit messen, um ggf. zu überprüfen, ob der Token noch da ist

- Was, wenn der Master selbst inaktiv ist?
  - Nach einer weiteren Zeitspanne ohne Kommunikation, gehen alle Agenten davon aus, dass der Master ausgefallen ist.
  - Der Nachfolger des alten Masters in der Liste übernimmt dessen Rolle

```
Liste:

~~ID 1 (Master)~~
ID 2 (Master)
ID 3
ID 4
```

- Und wenn auch das fehlschlägt?
  - Nach einem weiteren Timeout wird das Netz komplett neu initialisiert

Kommunikationssprachen

- Ein Agent *i* kann **keinen** Aufruf einer „Methode“ eines anderen Agenten *j* erzwingen, denn die Agenten sind **autonom**. Ebensowenig kann *i* interne Zustände (Variablen) von *j* direkt ändern
- Die Agenten können sich lediglich durch Kommunikation gegenseitig *beeinflussen*
- Wann ist es sinnvoll zu kommunizieren?
- Kann Agent j dem Agenten i glauben? Fehlfunktion? Eigene Bereicherung?? (naja, Sci-Fi...)

Sprechakte (Speech Acts)

- Begründer der Sprachakttheorie ist der Philosoph John Austin
  - Jeder Sprechakt korrespondiert mit einer „physikalischen Aktion“ bzw. löst eine Aktion aus
- Der Philosoph John Searle klassifiziert Sprechakte folgendermaßen
  - Representatives
    sagen, wie es sich verhält → behaupten, mitteilen, berichten
  - Directives
    zur Handlung/Unterlassung bewegen → bitten, befehlen, raten
  - Commissives
    sich selbst auf eine Handlung/Unterlassung festlegen
    → versprechen, vereinbaren, drohen
  - Expressives
    Ausdruck der eigenen Gefühlslage → danken, begrüßen, klagen
  - Declarations
    die Welt mit dem Gesagten verändern → ernennen, entlassen, verheiraten

KQML

- **Knowledge Query and Manipulation Language**
- Sprache zum Wissensaustausch
- „Envelope Format“ – Inhaltssprache wird ist uninteressant
- wurde 1993 als Standard vorgeschlagen
- lehnt sich an Sprechakt-Theorie an, um Nachrichten zu klassifizieren

```
(ask-one
   :content   coords(robot2,X,Y)
   :sender    robot1
   :receiver  robot2
   :language  PROLOG
   :ontology  Robot-Plain-Coords
)
```

In diesem Beispiel wird Prolog als Inhaltssprache verwendet.

Wooldridge schlägt *KIF* (Knowledge Interchange Format) vor

KQML

- Es ist egal, wie die einzelnen Agenten implementiert sind,
  Soll heißen: Die Agenten können sehr inhomogen sein
- Die verwendete Sprache (z.B. PROLOG) muss aber natürlich von allen beteiligten Agenten verstanden werden
- Es sind 40 „performatives“ definiert

```
(ask-one
   :content   coords(robot2,X,Y)
   :sender    robot1
   :receiver  robot2
   :language  PROLOG
   :ontology  Robot-Plain-Coords
)
```

performative

parameters

KQML - Performative

achieve	advertise	ask-about	ask-all	ask-if
ask-one	break	broadcast	broker-all	broker-one
deny	delete-all	delete-one	discard	eos
error	evaluate	forward	generator	insert
monitor	next	pipe	ready	recommend-all
recommend-one	recruit-all	recruit-one	register	reply
rest	sorry	standby	stream-about	stream-all
subscribe	tell	transport-address	unregister	untell

S will eine von R's Antworten zu C

S: Sender
R: Receiver
C: Content

VKB: virtual knowledge base

KQML - Performative

achieve	advertise	ask-about	ask-all	ask-if
ask-one	break	broadcast	broker-all	broker-one
deny	delete-all	delete-one	discard	eos
error	evaluate	forward	generator	insert
monitor	next	pipe	ready	recommend-all
recommend-one	recruit-all	recruit-one	register	reply
rest	sorry	standby	stream-about	stream-all
subscribe	tell	transport-address	unregister	untell

S will, dass R etwas in der Umwelt 'wahr macht'

S: Sender
R: Receiver
C: Content

VKB: virtual knowledge base

KQML - Performative

achieve	advertise	ask-about	ask-all	ask-if
ask-one	break	broadcast	broker-all	broker-one
deny	delete-all	delete-one	discard	eos
error	evaluate	forward	generator	insert
monitor	next	pipe	ready	recommend-all
recommend-one	recruit-all	recruit-one	register	reply
rest	sorry	standby	stream-about	stream-all
subscribe	tell	transport-address	unregister	untell

S fordert von R eine weitere Antwort zu einem vorhergehenden performative

S: Sender
R: Receiver
C: Content

VKB: virtual knowledge base

KQML - Performative

achieve	advertise	ask-about	ask-all	ask-if
ask-one	break	broadcast	broker-all	broker-one
deny	delete-all	delete-one	discard	eos
error	evaluate	forward	generator	insert
monitor	next	pipe	ready	recommend-all
recommend-one	recruit-all	recruit-one	register	reply
rest	sorry	standby	stream-about	stream-all
subscribe	tell	transport-address	unregister	untell

S kann keine informativere Antwort an R liefern

S: Sender
R: Receiver
C: Content

VKB: virtual knowledge base

KQML – Dialogbeispiel #1

ANFRAGE

```
(ask-one
   :sender   robot1
   :receiver robot2
   :content  coords(robot2,X,Y)
   :language PROLOG
   :ontology Robot-Plain-Coords
)
```

Roboter `robot1` will von `robot2` dessen Koordinaten wissen.

`X` und `Y` sind Prolog-Variablen, die für `robot1` unbekannt sind.

Durch die Angabe der Ontologie wird der Kontext exakt bestimmt.

ANTWORT

```
(tell
   :sender   robot2
   :receiver robot1
   :content  coords(robot2,12,38)
   :language PROLOG
   :ontology Robot-Plain-Coords
)
```

Roboter `robot2` antwortet mit dem Faktum, indem er die Variablen durch feste Werte ersetzt.

`tell` heißt eigentlich, dass der Sender den Inhalt in seiner VKB hat.

KQML – Dialogbeispiel #2

ANFRAGE

```
( stream-about
   :sender robot1
   :receiver robot2
   :language PROLOG
   :reply-with f1
   :content fire1
)
```

stream-about ermöglicht Multi-Antworten

reply-with legt einen Identifikator für die Antwort fest

Die Anfrage ist hier nicht PROLOG-konform

ANTWORT

```
(tell
   :sender robot2   :receiver robot1    :language PROLOG
   :in-reply-to f1  :content coords(fire1,12,38)
)
(tell
   :sender robot2   :receiver robot1    :language PROLOG
   :in-reply-to f1  :content intensity(fire1,5)
)
. . . .
(eos :sender robot2 :receiver robot1 :in-reply-to f1)
```

eos
End of Stream

FIPA ACL

Kritik an KQML

- Die Semantik ist in einigen Fällen unklar
- keine Möglichkeit, Commissives (Versprechen, Drohungen,...) auszudrücken
- zu viele Performative

FIPA agent communication language (1995)

- von der Foundation for Intelligent Physical Agents entwickelter Standard
- verbreitetet (z.B. verwendet in JADE)
- sehr ähnlich wie KQML
  (Envelope-Format, unabhängig von Content-Sprache, Syntax ähnlich)
- größter Unterschied zu KQML sind die Performative

FIPA ACL – Performative

accept-proposal	agree	cancel	cfp	confirm
disconfirm	failure	inform	inform-if	inform-ref
not-understood	propagate	propose	proxy	query-if
query-ref	refuse	reject-proposal	request	request-when
request-whenever	subscribe			

Passing information	Requesting information	Negotiation	Performing actions	Error handling

- Jedem Performativ ist eine Sematik mit Hilfe der Sprache **SL** zugewiesen
- In SL können Glaube (*belief*), Wünsche (*desires*) und unsicherer Glaube (*uncertain belief*) abgebildet werden.

FIPA ACL – Beispiel

```
(inform
   :sender    robot2
   :receiver  robot1
   :content   coords(robot2,12,38)
   :language  PROLOG
   :ontology  Robot-Plain-Coords
)
```

`inform` ist das Pendant zu `tell` aus KQML.

Der Unterschied ist, dass der hier der Sender sicher *glaubt*, dass der Inhalt auch wahr ist und sein Ziel ist es, dass auch der Empfänger den Inhalt *glaubt*.

FIPA ACL – Semantik

INFORM

Agent i möchte Agent j über φ informieren

Agent i glaubt, dass φ wahr ist

Agent j hat eine eindeutige „Meinung“: Entweder φ ist für ihn wahr oder falsch

$$\langle i, inform(j, \varphi) \rangle$$

feasibility precondition: $B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)$

rational effect: $B_j\varphi$

Agent j glaubt, dass φ wahr ist

Agent j ist sich nicht sicher, ob φ wahr oder falsch ist

Durch die Definition der Semantik kann überprüft werden, ob eine Nachricht gesendet werden soll oder darf. Der Agent kommuniziert dann also im Idealfall nur, wenn es Sinn macht.

FIPA ACL – Semantik

REQUEST

Agent i fordert Agenten j auf, eine Aktion α auszuführen

(leicht vereinfacht)

Agent i glaubt, dass Agent j der Agent ist, der die Aktion α ausführt

$$\langle i, request(j, \alpha) \rangle$$

feasibility precondition: $B_i Agent(\alpha, j) \wedge \neg B_i I_j Done(\alpha)$

rational effect: $Done(\alpha)$.

α wurde ausgeführt

Durch die Definition der Semantik kann überprüft werden, ob eine Nachricht gesendet werden soll oder darf. Der Agent kommuniziert dann also im Idealfall nur, wenn es Sinn macht.

Quellen

- Convington, Michael A.
  Speech Acts, Electronic Commerce, and KQML
  Artificial Intelligence Center – The University of Georgia
  Athens, Georgia 30602–7415 U.S.A.
  Revised 1997 June 9

- Bräunl, Thomas
  Embedded robotics –
  mobile robot design and applications with embedded systems
  3rd Edition S. 117 – 127
  2008 Springer Verlag, Berlin
  ISBN: 978-3-540-34318-9 , 3-540-34318-0

- Wooldridge, Michael J.
  An introduction to multiagent systems / Michael Wooldridge
  S. 163 – 199
  2002 John Wiley & Sons Ltd, West Sussex, England
  ISBN 0-471-49691-X

Quellen

- Internetquellen

  http://de.wikipedia.org/wiki/Sprechakttheorie
  http://de.wikipedia.org/wiki/KQML
  http://de.wikipedia.org/wiki/John_Searle