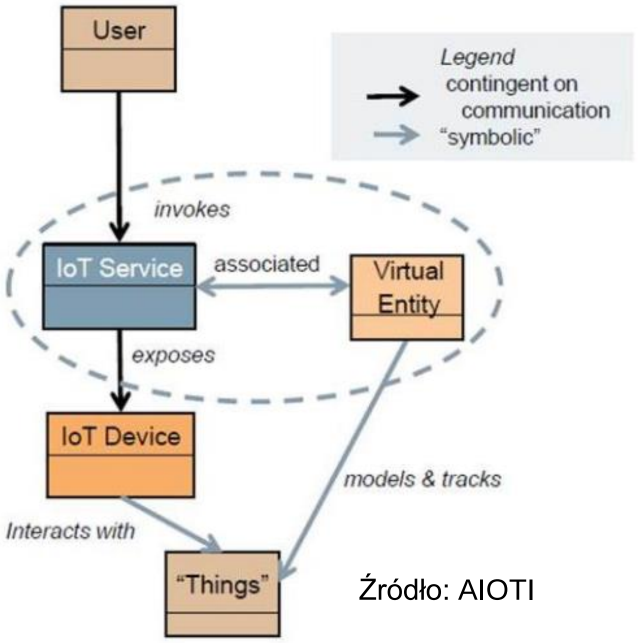
PSIR

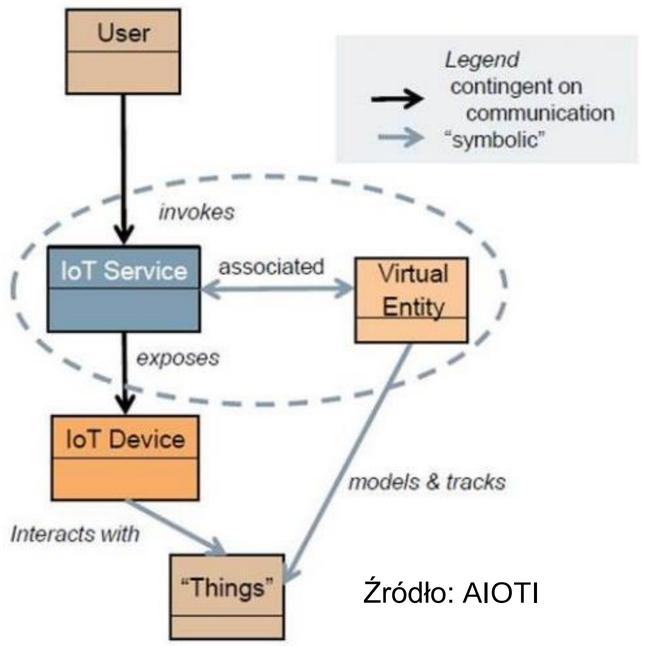
Wprowadzenie do projektu
2023Z: tuple space middleware for distributed computing

Jarosław Domaszewicz
Instytut Telekomunikacji Politechniki Warszawskiej



Źródło: AIOTI

PSIR project concept and history



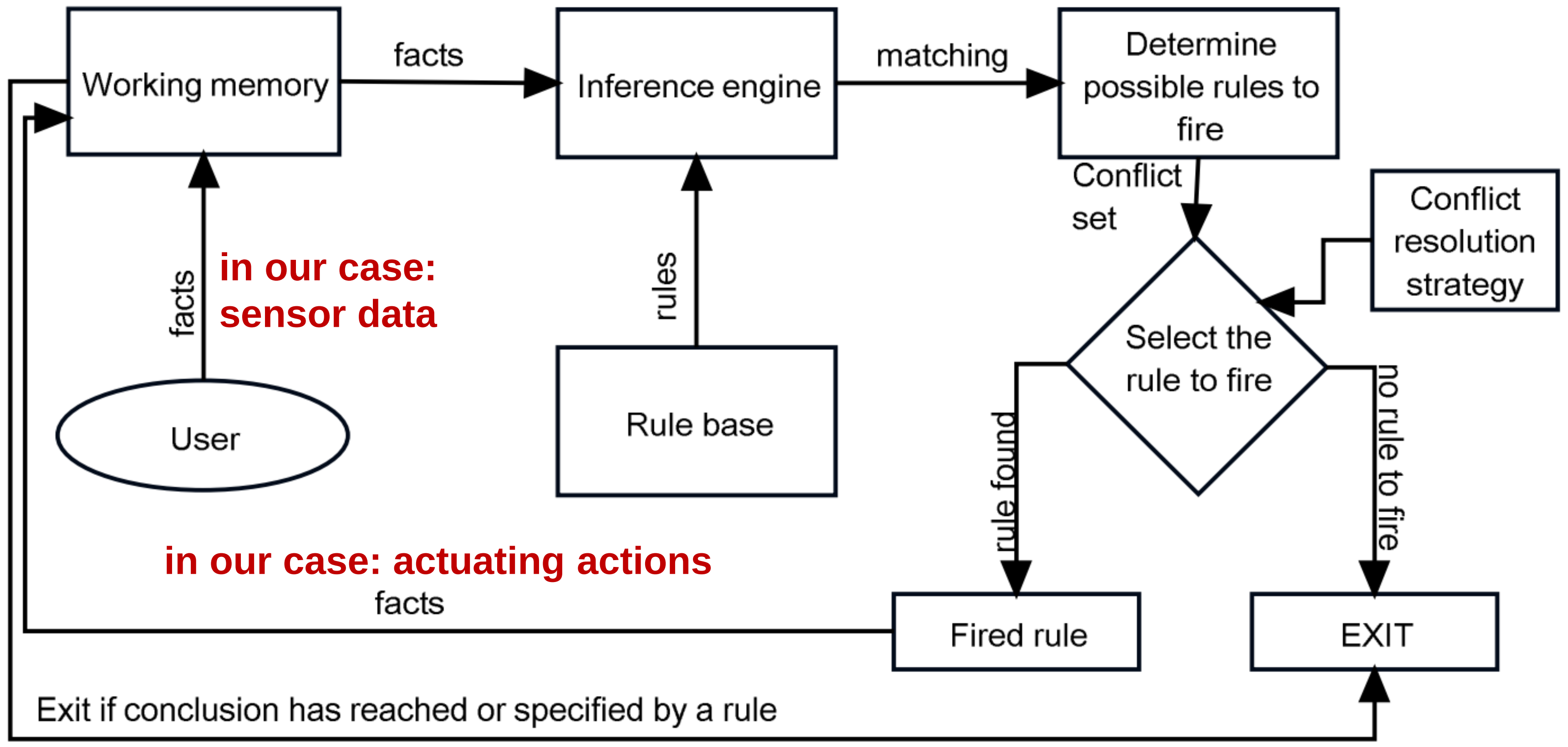
Źródło: AIOTI

PSIR PROJECT CONCEPT

A programming challenge that
helps develop solid programming skills
while being intellectually stimulating.

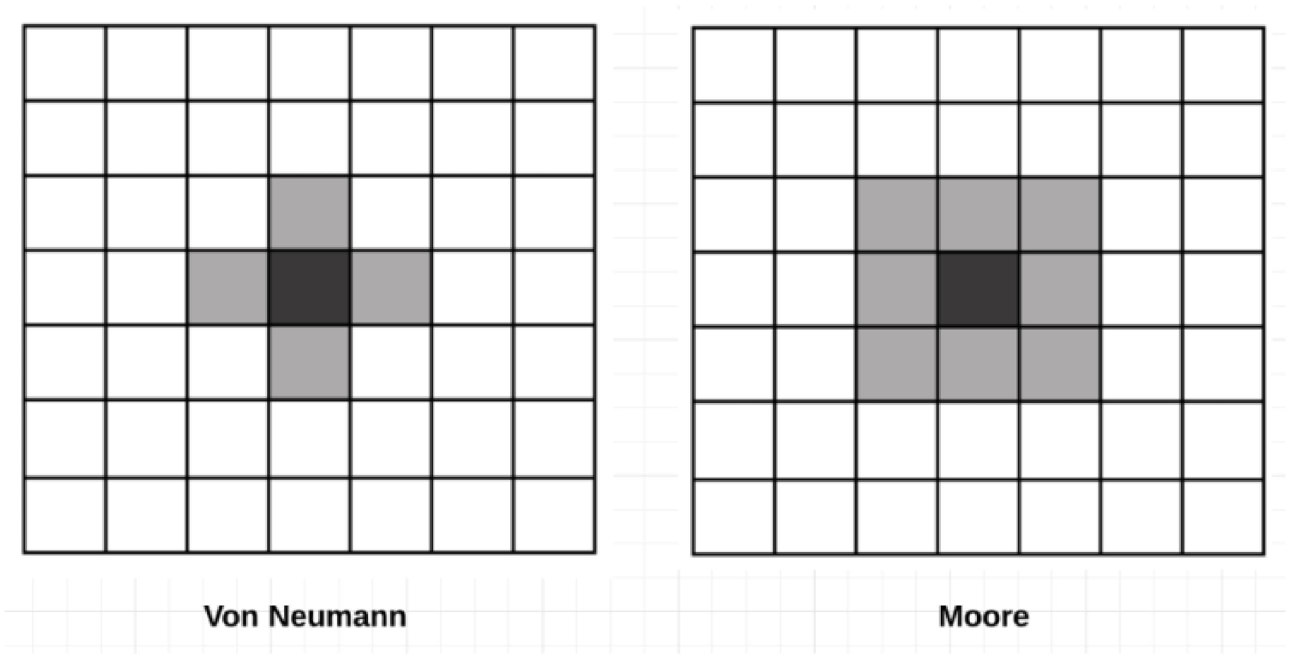
Enjoy stretching your mind!

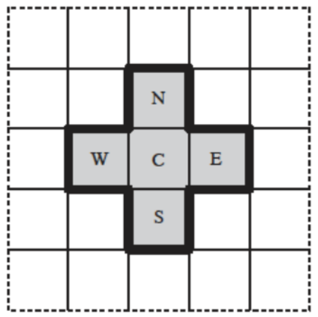
21Z: A SMART ENVIRONMENT RULE-BASED SYSTEM

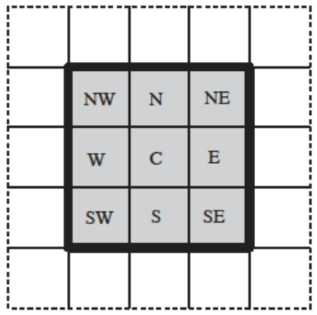


Source:
Intelligent Systems A Modern Approach,
Crina Grosan, Ajith Abraham,
Series: Intelligent Systems Reference Library, Springer,
DOI 10.1007/978-3-642-21004-4
Chapter 7, Rule-based Expert Systems, pp. 149-185

22L: Cellular automata with space partitioning





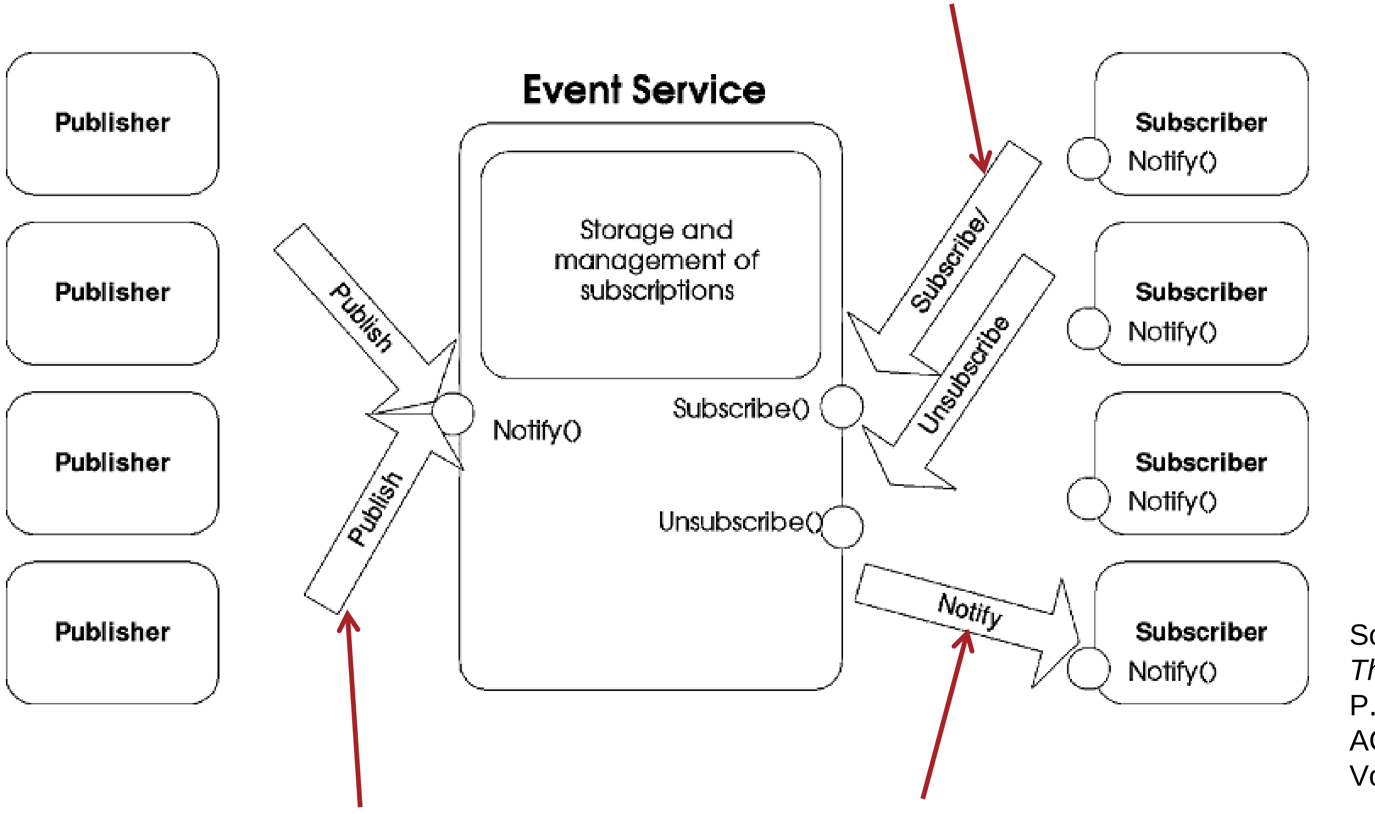


Source: Joseph Quartieri, Nikos E. Mastorakis, Gerardo Iannone, Claudio Guarnaccia
A Cellular Automata Model for Fire Spreading Prediction

Source: Washington Velasquez, Andres Munoz-Arcentales*y*, Thomas Michael Bohnertz, Joaquin Salvachua
Wildfire Propagation Simulation Tool using Cellular Automata and GIS

22Z: CONTENT-BASED PUBLISH/SUBSCRIBE

1. subscribers express interest in selected events



highly recommended

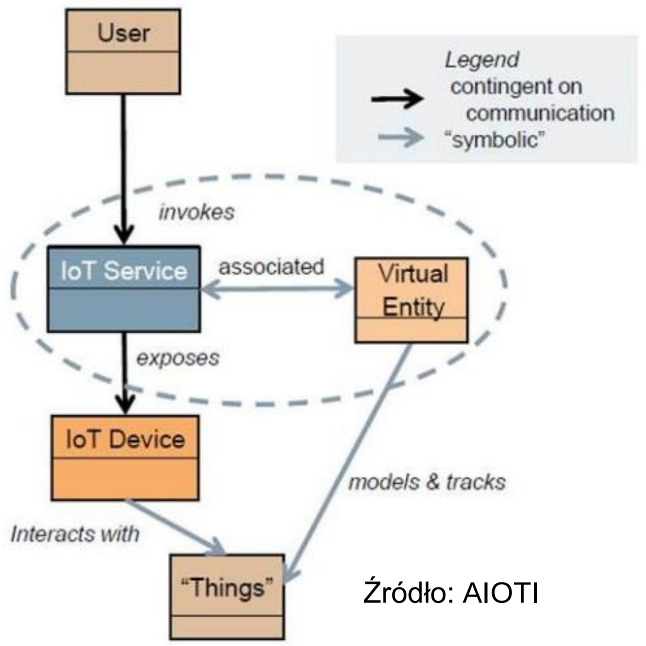
Source:
The Many Faces of Publish/Subscribe
P. Eugster et al.,
ACM Computing Surveys,
Vol. 35, No. 2, 2003

2. publishers produce assorted events

3. subscribers receive events they are interested in

- *publishers, subscribers, event service* (broker, server, middleware)
- note: push vs. pull, one-to-many, many-to-one

Clarification: the concept of middleware



Źródło: AIOTI

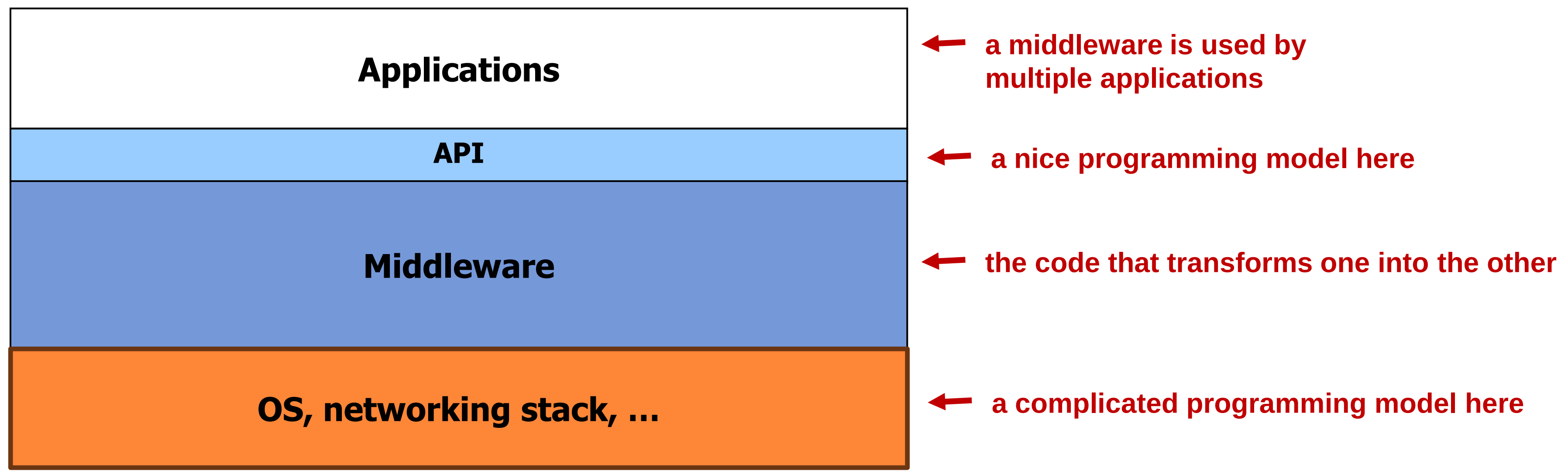
What is middleware?

In a distributed computing system,
middleware is defined as the software layer that
lies between the operating system and the applications
on each site of the system.

Its role is to make application development easier,
by providing common programming abstractions,
by masking the heterogeneity and the distribution
of the underlying hardware and operating systems,
and by hiding low-level programming details.

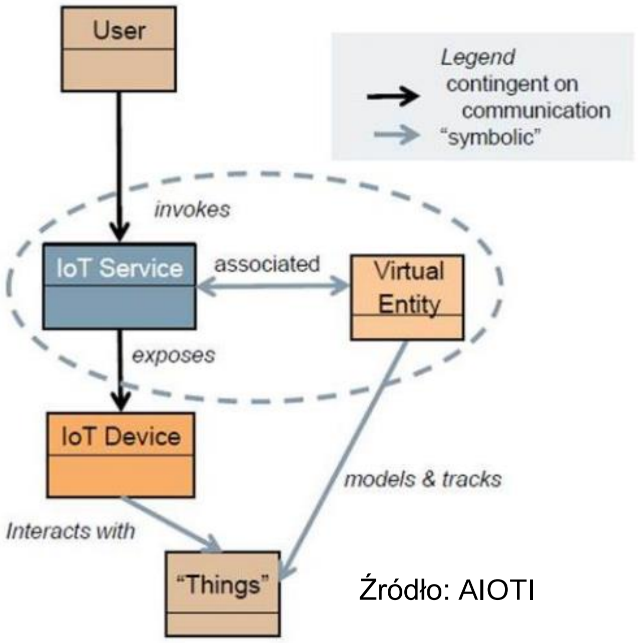
S. Krakowiak *Middleware Architecture with Patterns and Frameworks*, 2009

Middleware API


Applications
API
Middleware
OS, networking stack, …
a middleware is used by multiple applications
a nice programming model here
the code that transforms one into the other
a complicated programming model here

- API embodies a programmer-friendly programming model
  - usually, the level of abstraction is raised
- A middleware is to be used by multiple applications.
  - no point in developing one for a single application

Linda model / tuple space fundamentals



Źródło: AIOTI

WHAT IS OUR AREA?

- Consider a distributed or parallel application.
- It usually exists of multiple, concurrently running processes.
  - most often they run on different computers
- These processes need to coordinate their actions and communicate.
- How do they do that?
- There are several paradigms.
  - message passing
    - Remote Procedure Calls (RPC)
    - Message Passing Interface (MPI)
  - publish/subscribe
    - MQTT
  - …
  - tuple space
    - Linda

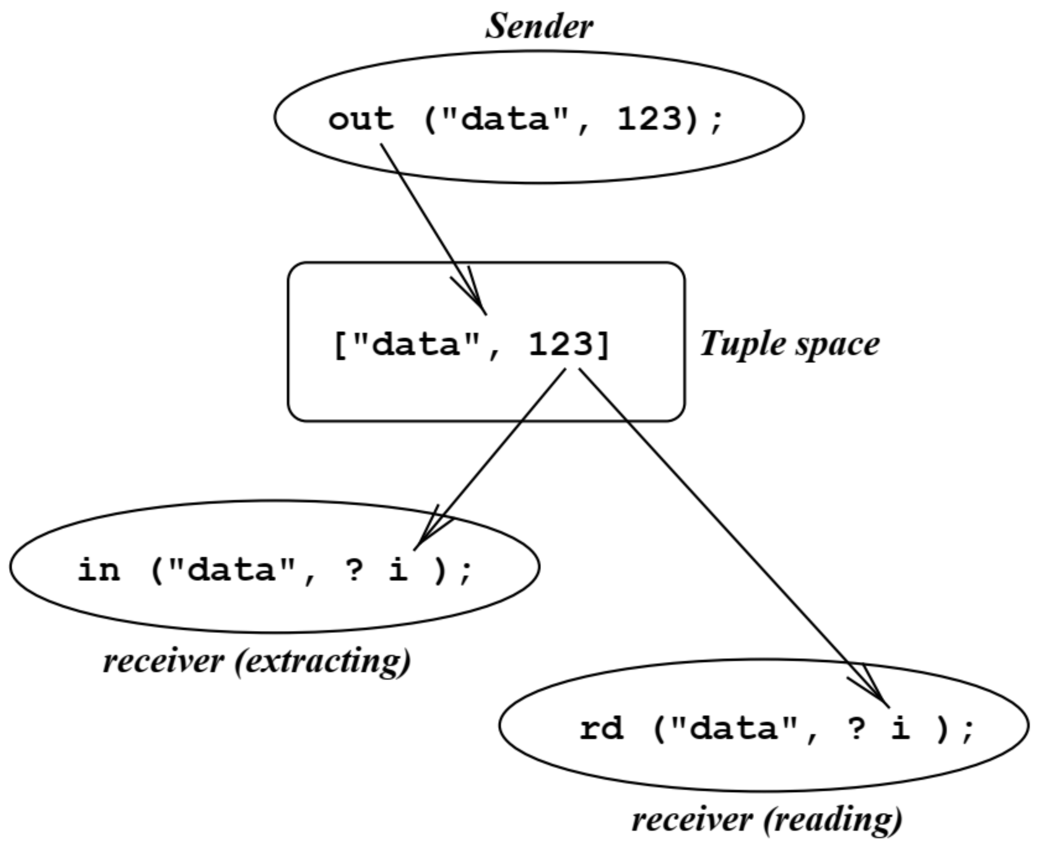
LINDA MODEL: TUPLE SPACE (1/2)

- A tuple space is associative, logically shared memory.
  - associative? see below
  - this is ”logical” sharing; there may be no physically shared memory

- Tuple space contains tuples.

- How processes coordinate their actions and communicate:
  - messages in Linda are never exchanged between two processes directly
  - a process with data to communicate adds it, as a tuple, to the tuple space
    - the tuple is not explicitly directed (addressed to) any specific process
  - a process that needs to receive data retrieves it, as a tuple, from the tuple space
    - the retrieved tuple may be removed from the tuple space or remain in it
  - note the high level of abstraction in the above description – that’s what we want

LINDA MODEL: TUPLE SPACE (2/2)



W. Hasselbring and M. Roantree. 1998.
A Generative Communication Service for Database Interoperability.
In Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (COOPIS '98). IEEE Computer Society, USA, 64–73.

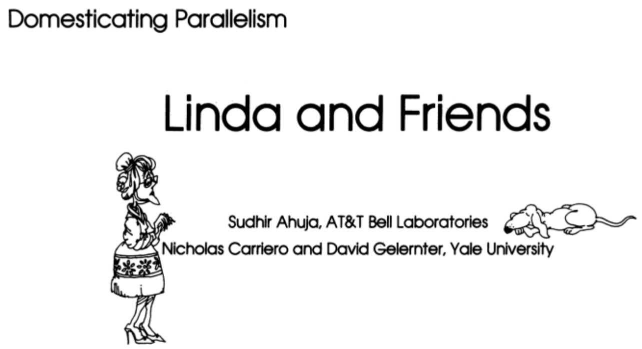
Decoupling is good!

read at home and think!

Advantages: Loose Coupling (Spatial)

Spatially, each process in a parallel program will usually develop a result or a series of results that will be accepted as input by certain other processes. Uncoupling suggests that process *q* should not be required to know or care that process *j* accepts *q*'s data as input.

Instead of requiring *q* to execute an explicit "send this data to *j*" statement, we would rather that *q* be permitted simply to tag its new data with a logical label (for example, "new data from *q*") and then forget about it, under the assumption that any process that wants it will come and get it. At some later point in program execution, a different process may decide to deal with *q*'s data. Under the spatially-uncoupled scheme, this won't matter to *q*.



Domesticating Parallelism

Linda and Friends

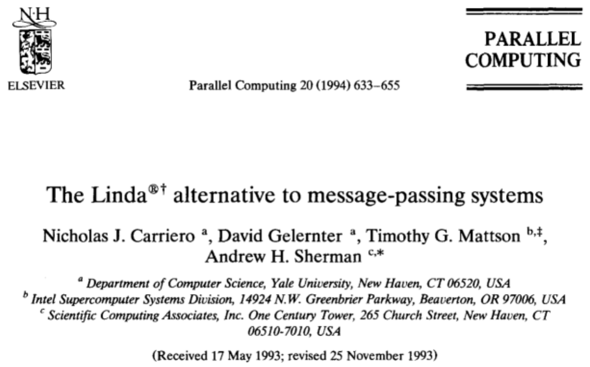
Sudhir Ahuja, AT&T Bell Laboratories

Nicholas Carriero and David Gelernter, Yale University

Decoupling is good!

ADVANTAGES: LOOSE COUPLING (TEMPORAL)

Linda's loose interprocess coupling has other advantages as well. Tuple space can be viewed as a long term data memory - once installed, tuples remain in tuple space until they are explicitly removed by some process. Thus, processes can interact through time as well as space (or machine location), since the producer and consumer of a tuple need never coexist simultaneously.


ELSEVIER

Parallel Computing 20 (1994) 633–655

PARALLEL COMPUTING

The Linda®† alternative to message-passing systems

Nicholas J. Carriero [a], David Gelernter [a], Timothy G. Mattson [b,‡], Andrew H. Sherman [c,*]

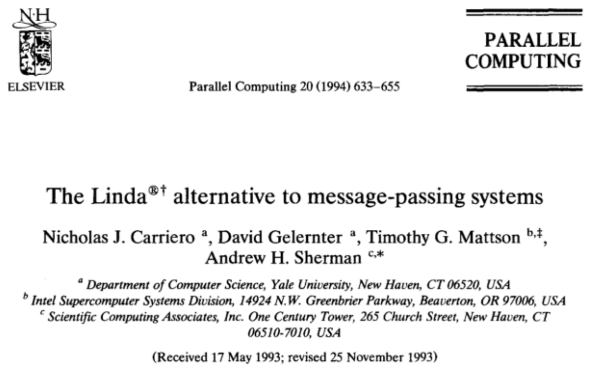
[a] *Department of Computer Science, Yale University, New Haven, CT 06520, USA*
[b] *Intel Supercomputer Systems Division, 14924 N.W. Greenbrier Parkway, Beaverton, OR 97006, USA*
[c] *Scientific Computing Associates, Inc. One Century Tower, 265 Church Street, New Haven, CT 06510-7010, USA*

(Received 17 May 1993; revised 25 November 1993)

Decoupling is good!

ADVANTAGES: LOOSE COUPLING

- ”Each individual process can be developed more-or-less independently of the others.”


ELSEVIER

Parallel Computing 20 (1994) 633–655

PARALLEL COMPUTING

The Linda®† alternative to message-passing systems

Nicholas J. Carriero [a], David Gelernter [a], Timothy G. Mattson [b,‡], Andrew H. Sherman [c,*]

[a] *Department of Computer Science, Yale University, New Haven, CT 06520, USA*
[b] *Intel Supercomputer Systems Division, 14924 N.W. Greenbrier Parkway, Beaverton, OR 97006, USA*
[c] *Scientific Computing Associates, Inc. One Century Tower, 265 Church Street, New Haven, CT 06510-7010, USA*

(Received 17 May 1993; revised 25 November 1993)

TUPLE

- A tuple is an ordered sequence of data (of fields).
- Each field of a tuple contains actual data of one of valid types.
- The number of fields and their types may vary.
- It is common (but not essential) to have a string as the first field.
  - it serves as a descriptive tag for the tuple
  - we’ll stick to this convention
- A special field may be used as a unique application identifier.
  - so that there is no interference between applications concurrently using the tuple space

- Examples:

`(”temperature”, 22.5, CELSIUS)`	**a sensor reading**
`(”check_if_prime”, 27)`	**a task**
`(”is_prime”, 17)`	**a result**
`(„new_pin_value”, 8, OFF)`	**a command**

INSERTING TUPLE INTO TUPLE SPACE: `out()`

- `out(t)`
  - a tuple `t` is added to the tuple space
  - the executing process continues immediately
- Example
  - `out(”temperature”, temp, unit);` **`temp` and `unit` are variables**
  - `out(”temperature”, 22.5, CELSIUS);` **`CELSIUS` has been `#define’d`**

Tuple space as associative memory

- This is about retrieving tuples from the tuple space.
- Tuple space is an associative memory.
  - tuples have no addresses
  - they are selected for retrieval based on any combination of their field values
- Example:
  - a five-element tuple (A, B, C, D, E) may be referenced as

  - ”the five-element tuple whose first element is A” …
  - `(A, ?w, ?x, ?y, ?z)`

  - … or as ”the five-element tuple whose second element is B and fifth is E” …
  - `(?v, B, ?x, ?y, E)`

  - … or by any other combination of element values

TEMPLATE

- Template
  - a template is a sequence of typed fields that may be either actual values (just as in tuples), or formal place-holders

- A tuple t matches a template s if
  - both have the same number of fields, and …
  - … the types of the fields match pairwise, and …
  - … each actual value in s matches the value in the corresponding field of t

- Example:
  - consider `(”nice_constants”, 128, 3.14)`, assume 3.14 is of type float
  - it matches the templates
    - `(”nice_constants”, 128, ? float)`
    - `(”nice_constants”, ? int, ? float)`
  - it doesn’t match the templates
    - `(”nice_constants”, 128, ? float, ?int)`
    - `(”nice_constants”, ? float, ? float)`
    - `(”nice_constants”, 129, ? float)`

Retrieving tuple from tuple space: `in()`,`inp()`

- `in(s)`
  - `s` is a template
  - a tuple `t` that matches the template `s` is withdrawn from the tuple space
  - the actual values in t are assigned to the formal placeholders in s
  - the executing process continues
  - if no matching tuple is available when `in(s)` executes, the executing process blocks until one becomes available, then proceeds as before
  - if many matching tuples are available, one is chosen arbitrarily

- `inp(s)`
  - just like `in()`, but it doesn't block
  - a return value indicates success or failure

- Example
  - `in(”temperature”, ? int, CELSIUS);` **this is not C!**
  - `(”check_if_prime”, ? int);`

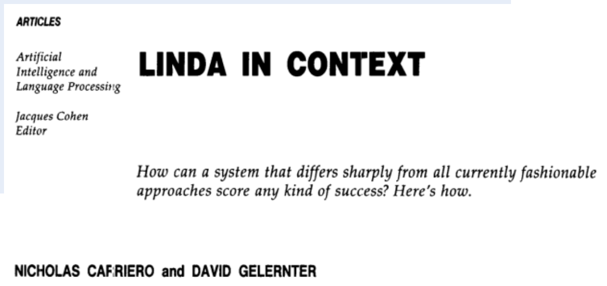
RETRIEVING TUPLE FROM TUPLE SPACE: `RD()`, `RDP()`

- `rd(s)`
  - works as `in(s)`, except that …
  - … the matched tuple remains in the tuple space
- rdp(s)
  - just like `rd()`, but it doesn't block
  - a return value indicates success or failure

EXAMPLE: CLIENT-SERVER IN LINDA

This is pseudo-code (”?” not in C).

```
server()
{
    int index = 1;
    . . .
    while (1) {
        in("request", index, ? req);
        . . .                         // generate a response
        out("response", index++, response);
    }
}

client()
(
    int index;
    . . .
    in("server index", ? index);  // get an index for your request
    out("server index", index+l); // index for the next request
    . . .
    out(”request", index, request);
    in("response", index, ? response);
    . . .
}
```



ARTICLES

Artificial Intelligence and Language Processing

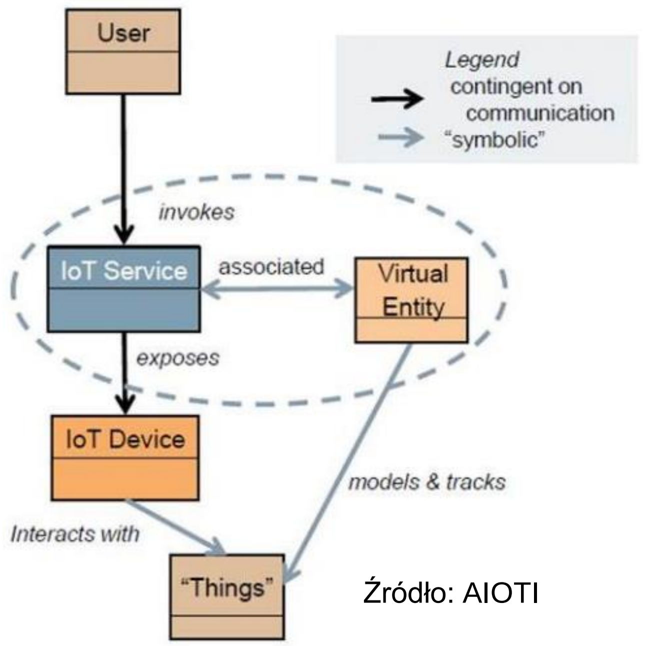
Jacques Cohen Editor

LINDA IN CONTEXT

How can a system that differs sharply from all currently fashionable approaches score any kind of success? Here's how.

NICHOLAS CARRIERO and DAVID GELERNTER

Tuple space in C



Źródło: AIOTI

ADDING A TUPLE SPACE TO A LANGUAGE

- A tuple space is added to a base programming language (e.g., C) as a language extension or a library.

- A language extension (adding new syntax and semantics to a language) requires a preprocessor …
  - a preprocessor would transform source code in an extended language into source code in a standard language

- … so our choice is a library with a well defined API
  - you can consider a library a kind of (limited) middleware

Representing Linda model via C API

- There is no "one and only correct" API.
  - the details of API may change as long as we stick to the underlying application layer protocol (see below)
- Below you can find an API that you should use in the project.
  - the API has some benefits when it comes to mastering C

TUPLE_SPACE.H (HEADER FILE WITH API)

this is what a typical header file looks like

```c
#ifndef TUPLE_SPACE_H
#define TUPLE_SPACE_H

#define TS_YES          1
#define TS_NO           0
#define TS_INT          0
#define TS_FLOAT        1
#define TS_SUCCESS      1
#define TS_FAILURE      0
```

typedef, section 6.7 of K&R 2nd ed.

```c
typedef struct {
      int       is_actual;          /* does the data member contains data */
      int       type;               /* what is the type of the data member */
      union {
```

unions, section 6.8 of K&R 2nd ed.

```c
               int int_field;
               float float_field;
      }        data;
} field_t;              /* a new type corresponding to one field of a tuple */

/* API */
/* these functions return TS_SUCCESS or TS_FAILURE */
/* parameters: tuple name, other fields, no. of other fields */
int ts_out(char*, field_t*, int);
int ts_inp(char*, field_t*, int);
int ts_rdp(char*, field_t*, int);
#endif
```

USING THE API (1/2)

```
#include ”tuple_space.h”
int main(void)
{
    int nice_power;
    double temp, pi;
    field_t my_tuple[2];          /* an array of fields (name not included) *
    field_t my_template[1];

    /* make a tuple */
    my_tuple[0].is_actual = YES
    my_tuple[0].type = INT_FIELD;
    my_tuple[0].data.int_field = 128;
    my_tuple[1].is_actual = YES
    my_tuple[1].type = FLOAT_FIELD;
    my_tuple[1].data.float_field = 3.14;
    /* add a tuple to the tuple space */
    ts_out("nice_constants", my_tuple, 2); /* ("nice_constants",128,3.14) */
```

Include the header file before using the API

the way to access a union member

the number of elements in the array

passing a pointer to the first element of an array (you may want to read Section 5.3 *Pointers and Arrays* of K&R 2nd ed., or equivalent)

passing a pointer to the first character of the string (you may want to read Section 5.5 *Character Pointers and Functions* of K&R 2nd ed., or equivalent)

Using the API (2/2)

```
    /* make a template */
    my_template[0].is_actual = NO;
    my_template[0].type = FLOAT_FIELD;      /* need to specify the type */
    /* retrieve and remove a tuple with temperature */
    /* some other process must have produced a tuple matching the template */
    ts_inp("temperature", my_template, 1); /* ("temperature",?float) */
    temp = my_template[0].data.float_field;

    /* transform a previously used tuple into a template */
    my_tuple[0].is_actual = NO;
    my_tuple[1].is_actual = NO;
    ts_rdp("nice_constants", my_tuple, 2); /* ("nice_constants",?int,?float) */
    nice_power = my_tuple[0].data.int_field;/* 128 – from the tuple space */
    pi = my_tuple[1].data.float_field;      /* 3.14 – from the tuple space */

    return 0;
}
```

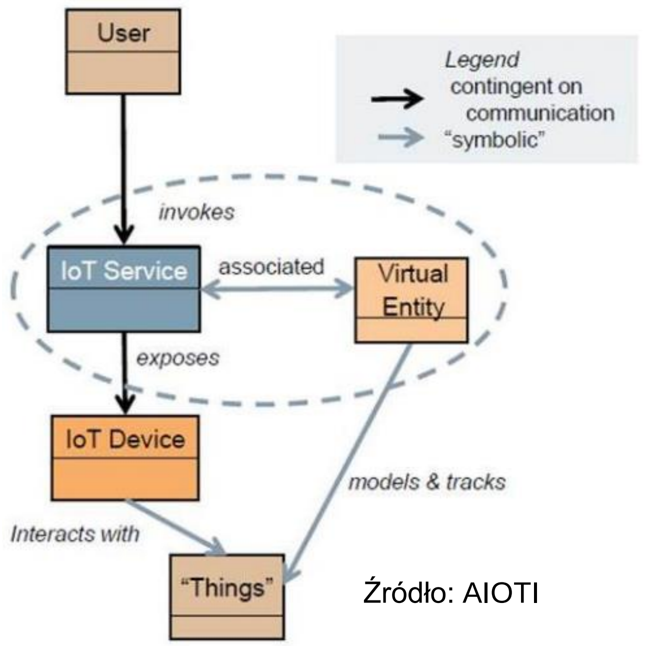
USING THE API (2/2)

```
    /* make a template */
    my_template[0].is_actual = NO;
    my_template[0].type = FLOAT_FIELD;     /* need to specify the type */
    /* retrieve and remove a tuple with temperature */
    /* some other process must have produced a tuple matching the template */
    ts_inp("temperature", my_template, 1); /* ("temperature",?float) */
    temp = my_template[0].data.float_field;

    /* transform a previously used tuple into a template */
    my_tuple[0].is_actual = NO;
    my_tuple[1].is_actual = NO;
    ts_rdp("nice_constants", my_tuple, 2); /* ("nice_constants",?int,?float) */
    nice_power = my_tuple[0].data.int_field;/* 128 – from the tuple space */
    pi = my_tuple[1].data.float_field;     /* 3.14 – from the tuple space */

    return 0;
}
```

What's wrong with this code?

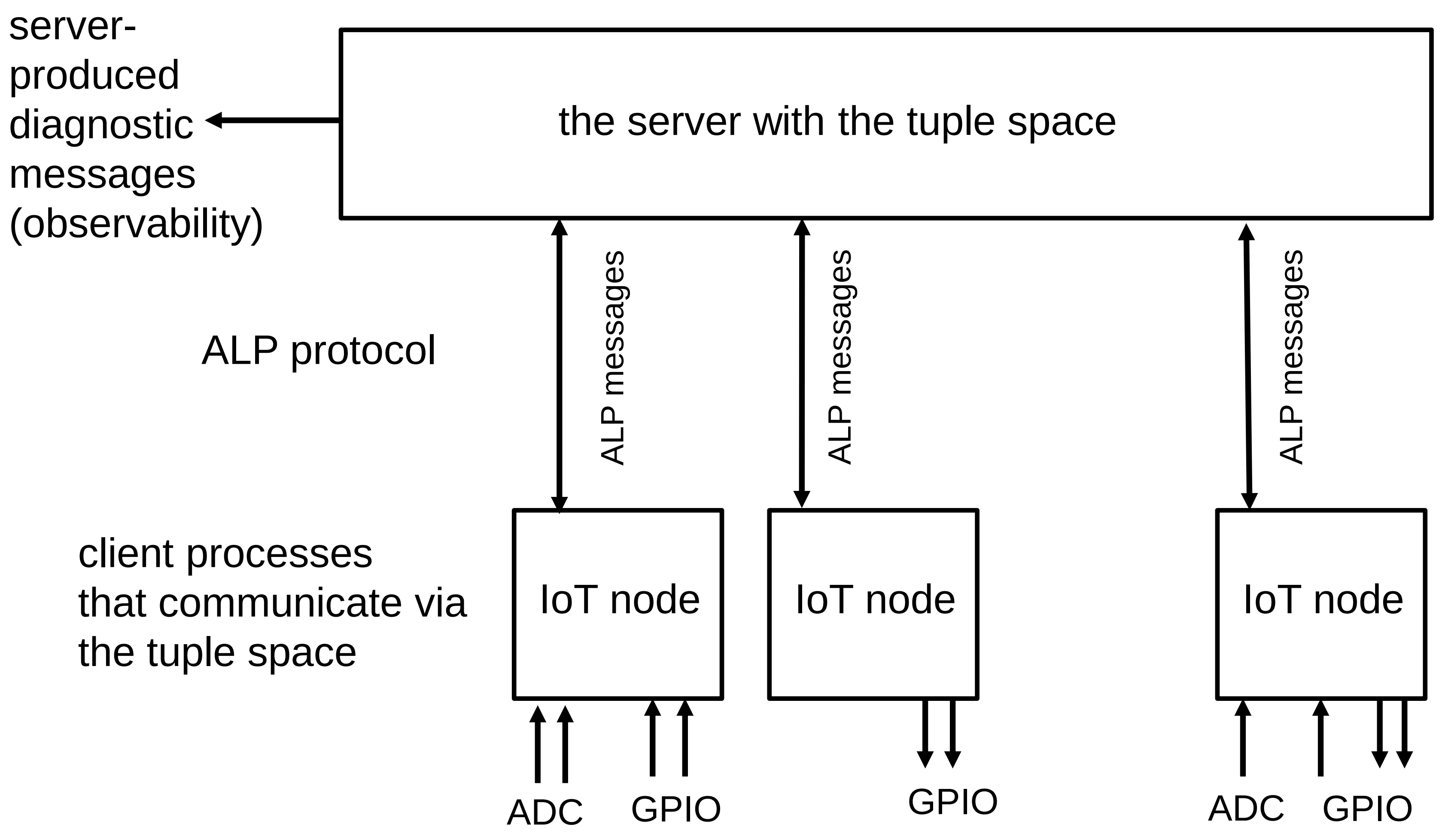
The return values not checked!
Big mistake; don't do this!

System high-level architecture

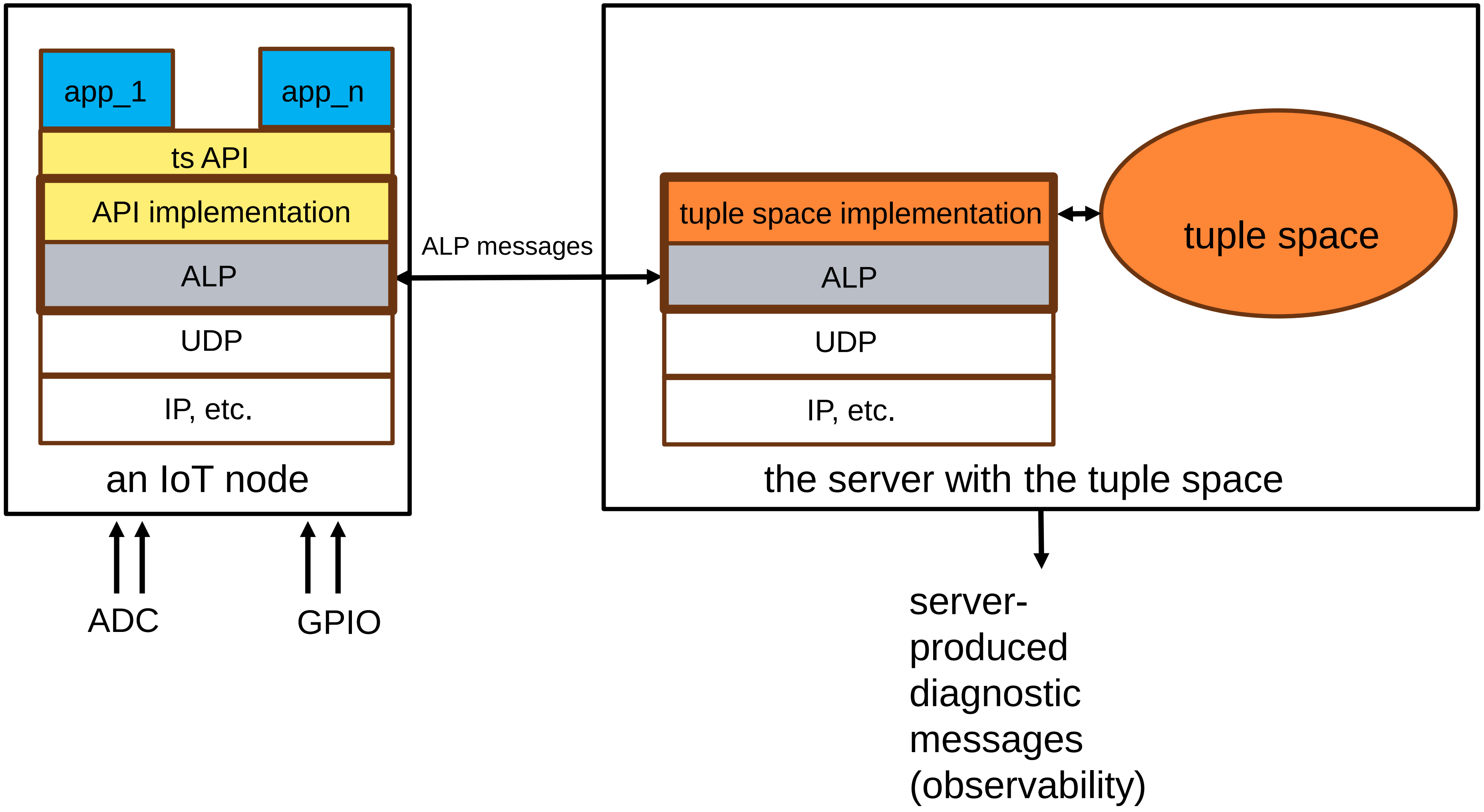


Źródło: AIOTI

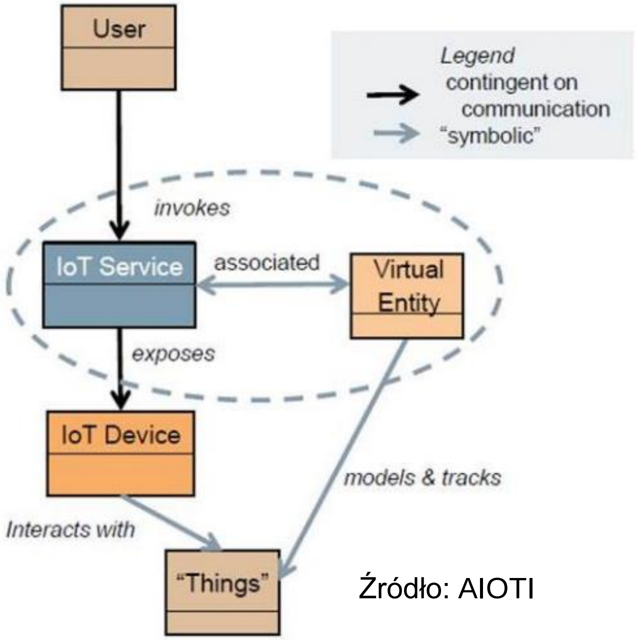
SYSTEM ARCHITECTURE


server-produced diagnostic messages (observability)
the server with the tuple space
ALP protocol
ALP messages
ALP messages
ALP messages
client processes that communicate via the tuple space
IoT node
IoT node
IoT node
ADC
GPIO
GPIO
ADC
GPIO

SYSTEM ARCHITECTURE


app_1
app_n
ts API
API implementation
ALP
UDP
IP, etc.
an IoT node
ADC
GPIO
ALP messages
tuple space implementation
ALP
UDP
IP, etc.
tuple space
the server with the tuple space
server-produced diagnostic messages (observability)

ALP protocol



Źródło: AIOTI

ALP

- An application layer protocol.
- Used for communication between a client and a server maintaining a tuple space.
- It's entirely your creation.
- Its purpose is to enable tuple space-based communication and coordination.
- It should be language and platform independent.
  - e.g., clients based on different platforms, written in different languages may participate in a distributed application
  - btw, this is nothing new: making it possible for participating parties to be heterogeneous is the benefit of having a well specified protocol
    (and one of the main reasons to specify it)

ALP IS A BINARY PROTOCOL (1/2)

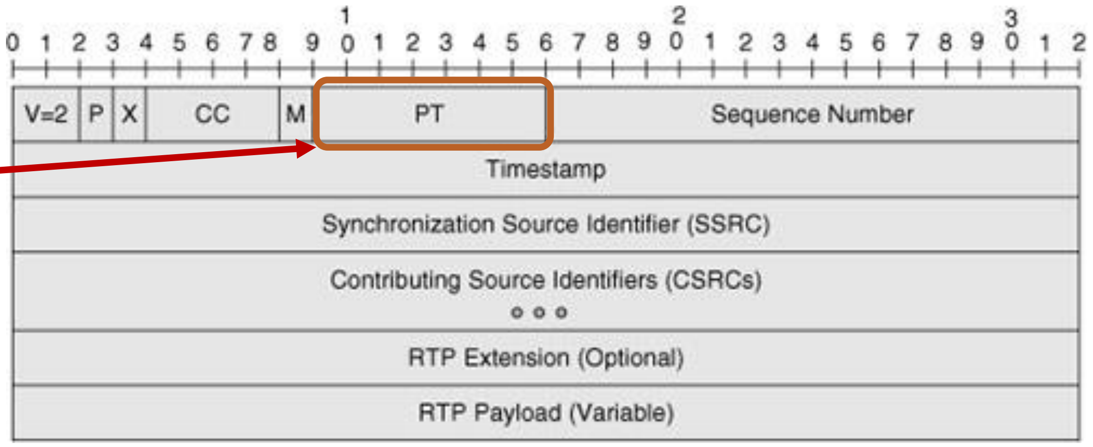
- „zaprzyjaźnij się z bajtem” (*befriend the byte*)
- ALP should be a *binary* protocol
- why? to save memory and the amount of transmitted data
- you need to define message formats at the bit level
  - identify bit fields within words
  - for each bit field, specify encodings (meanings of different bit patterns)

ALP IS A BINARY PROTOCOL (2/2)

- example: RTP (Real-time Transport Protocol)
  - a binary application layer protocol to transfer, e.g., voice in „phone calls” (VoIP)

bit number

bit field
Payload Type (PT):
a 7-bit field that identifies
the format of the RTP payload



- you may also look at RFC 791 (Fig. 4) or RFC 793 (Fig. 3)
- hint: when working with a binary protocol, make sure you understand things like:
  - network byte order
  - endianness

document the format
of ALP messages like this

ALP IS A BINARY PROTOCOL (3/3)

- another example: IPv4 (RFC 791)

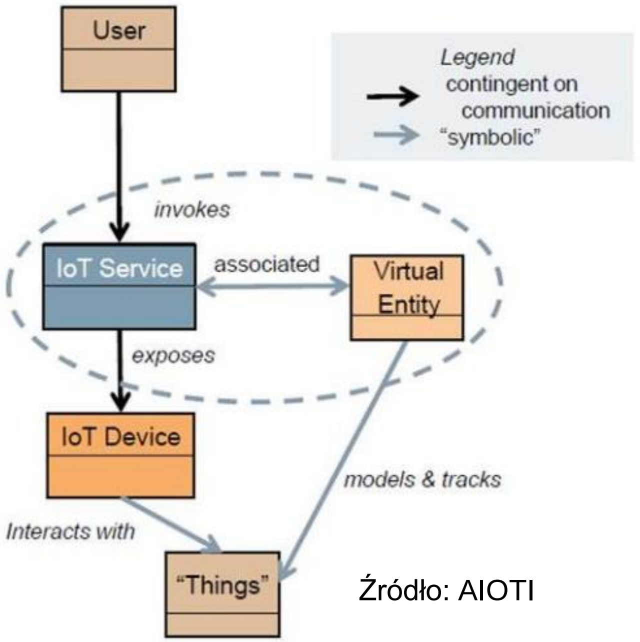
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                 Example Internet Datagram Header

                            Figure 4.
```

- in the case of ALP, you also need to specify the format of the payload
- you may also look at RFC 793 (Fig. 3)

ALP SHOULD RUN ON TOP OF UDP

- UDP often used in IoT due to its simplicity
- unreliable
  - add a simple reliability scheme
  - transmit, wait for ACK, if no ACK, retransmit

Server



Źródło: AIOTI

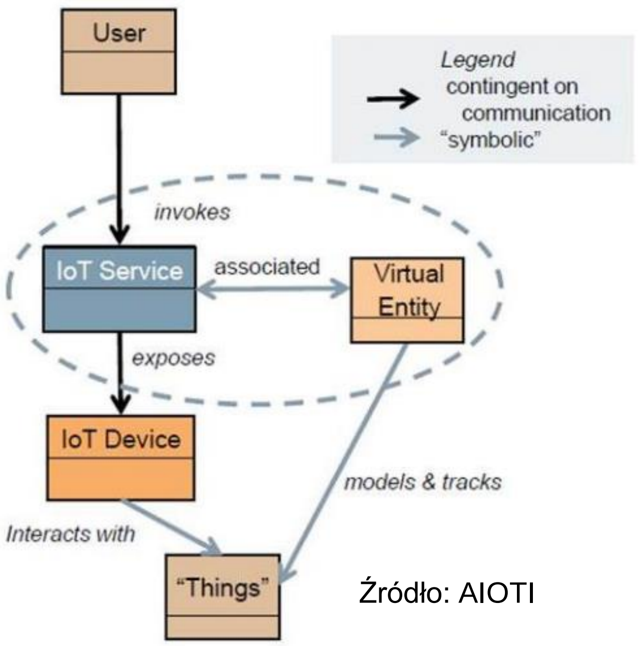
SERVER FUNCTIONALITY

- Receives, processes, and sends ALP messages.
- Maintains the tuple space.
  - note: the tuple space does not belong to any single application
- Updates metrics and produces diagnostic messages.
  - examples:
    - the number of all messages received so far
    - the number of OUT messages received so far
    - the number of INP and RDP messaged messages received so far
    - the number of tuples in the tuple space
    - average message length

SERVER PLATFORM

- Use Linux system API and C language facilities only
  - no libraries, no platforms, no middleware, …

Nodes



Źródło: AIOTI

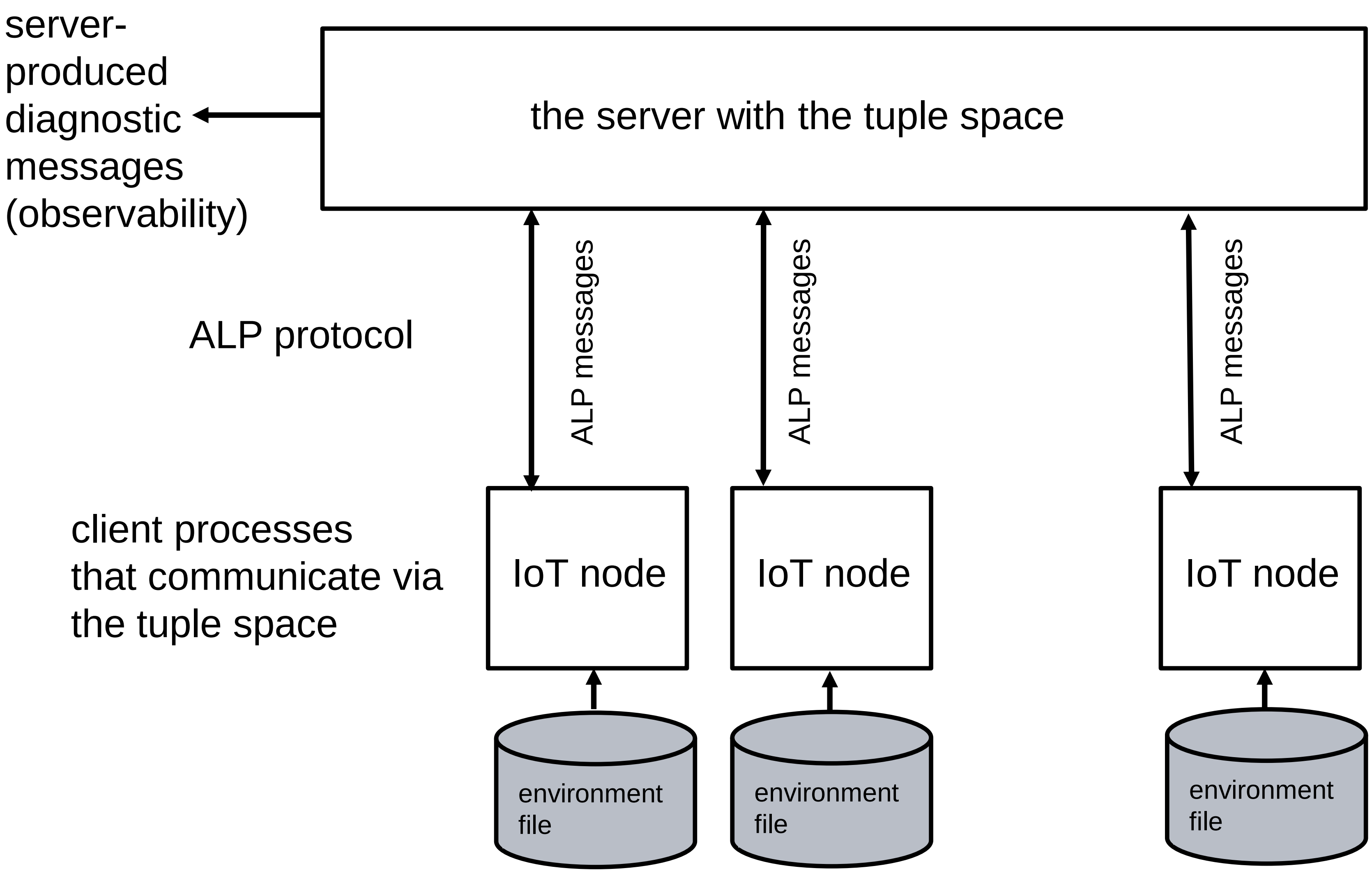
Node functionality

- Receives, processes, and sends ALP messages.
- Implements the tuple space API.
- Implements the application logic.

Node platform

- Arduino emulator
- Use ”standard” Arduino API and PSIR extensions only.
- The emulator supports UDP only.

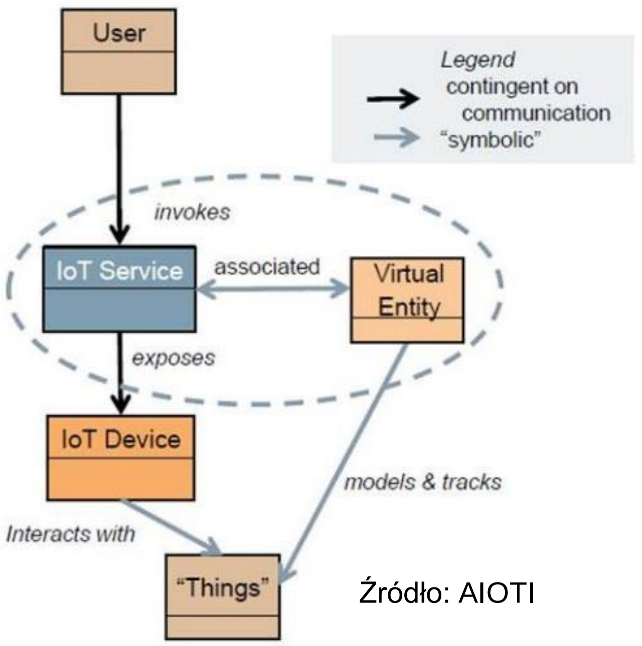
ENVIRONMENT FILES


server-produced diagnostic messages (observability)
the server with the tuple space
ALP protocol
ALP messages
ALP messages
ALP messages
client processes that communicate via the tuple space
IoT node
IoT node
IoT node
environment file
environment file
environment file

Environment file

```
+ qTemperature,quantity,Z0    # input 0=-10 1023=+20
+ qHumidity,quantity,Z1
+ sOpening,status,D1          # input 0=OPEN, 1=CLOSED
+ aSwitch,action,D2           # output 0=ON, 1=OFF


: 1000,qAirTemperature, 20
: 3000,qAirTemperature, 21
: 5000,sOpening, 0
: 7000,qHumidity, 512
```

Applications



Źródło: AIOTI

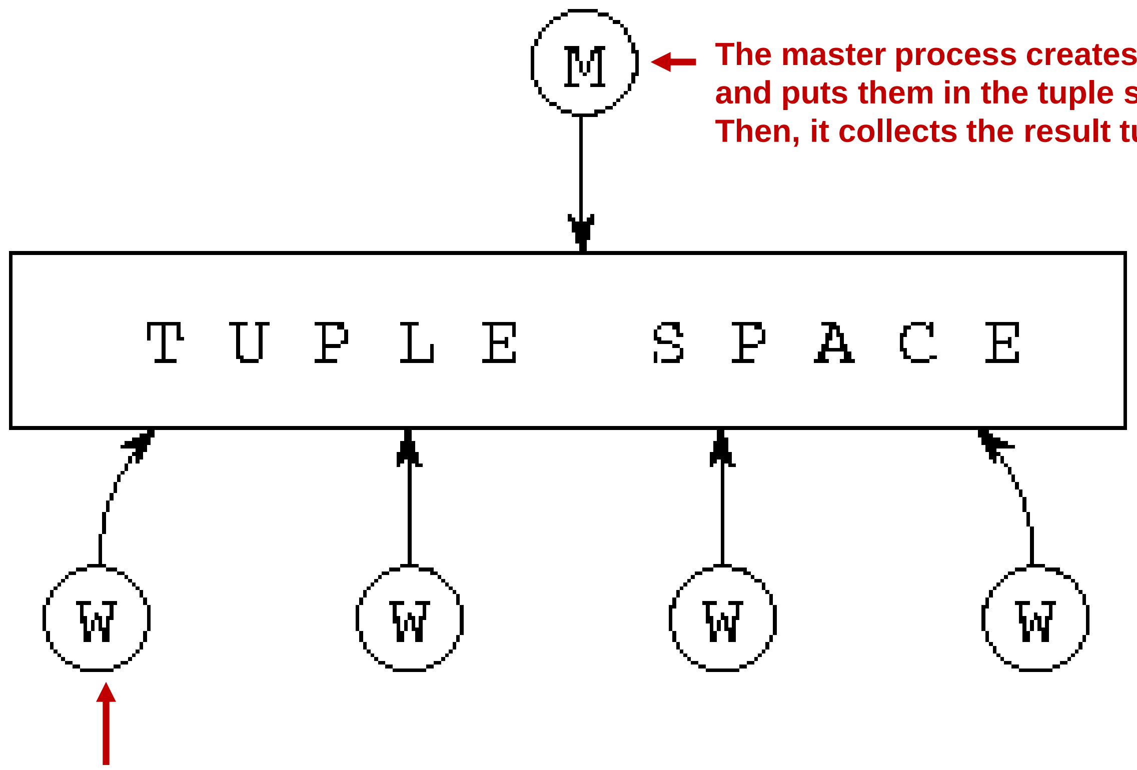
WHY APPLICATIONS?

- The applications are deliberately <u>very</u> simple.
  - <u>your primary outcome is the middleware, not applications</u>
- Two applications exemplify the concept that a middleware is built to ease the development of multiple applications.
- You can use the apps to test your implementation of the tuple space middleware.
- You will use the apps to demonstrate the middleware.

Application 1 (1/2)

- This app is meant to illustrate one way to structure a distributed application.
  - the master/worker model
- You may replace this app with another one, of similar structure and complexity.
  - if so, keep the master/worker model

MASTER/WORKER MODEL



The master process creates task tuples and puts them in the tuple space. Then, it collects the result tuples.

A worker process retrieves a task tuple, does the task computation, and puts a result tuple in the tuple space. Then, it retrieves another task tuple.

Source: Ian Foster, *Designing and Building Parallel Programs*
https://www.mcs.anl.gov/~itf/dbpp/text/node44.html

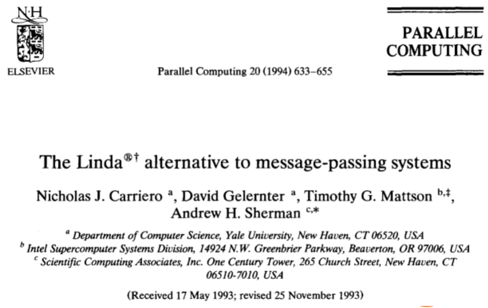
read at home and think!

MASTER/WORKER MODEL AND LOAD BALANCING

There are many situations, however, where static load balancing strategies fail because it is impossible to create an even division of labor based on a priori analysis. Therefore, it is important that Linda can efficiently support dynamic load balancing strategies as well as static ones. One technique for doing so involves viewing tuple space as a 'bag' of tasks to be performed, with individual tuples holding the inputs for a single task. Processes can acquire one of these 'task tuples,' perform the required work, and create a new tuple containing the results.

Load balance occurs almost automatically, even with heterogeneous processors, since processes that complete tasks quickly can complete several tasks in the time taken by other processes to complete just one.

The key to the efficiency of this approach is that there need be no a priori assignment of tasks to processes; the Linda operations implicitly support the notion that processes can acquire task tuples exactly as rapidly as they are ready for them.



N·H ELSEVIER

Parallel Computing 20 (1994) 633–655

PARALLEL COMPUTING

The Linda®† alternative to message-passing systems

Nicholas J. Carriero [a], David Gelernter [a], Timothy G. Mattson [b,‡], Andrew H. Sherman [c,*]

[a] *Department of Computer Science, Yale University, New Haven, CT 06520, USA*
[b] *Intel Supercomputer Systems Division, 14924 N.W. Greenbrier Parkway, Beaverton, OR 97006, USA*
[c] *Scientific Computing Associates, Inc. One Century Tower, 265 Church Street, New Haven, CT 06510-7010, USA*

(Received 17 May 1993; revised 25 November 1993)

Application 1 (2/2)

- The manager process:
  - outputs the tuple (”check_if_prime”, n), for n=2,…, N
  - this is a task to be performer by a worker process
  - inputs the template (”is_prime”, ?n) and template (”is_not_prime”, ?n) until it collects N-1 tuples
  - prints all the numbers, with info on whether they are primes or not

- The worker process:
  - inputs the template (”check_if_prime”, ?n)
  - checks if n is prime
    - do it any way you want,
    - need not be efficient
  - outputs the tuple (”is_prime”, ?n) and tuple (”is_not_prime”, ?n)

- At least two instances of the worker process should be created.

APPLICATION 2 (1/2)

- This application is meant to illustrate the use of sensing in IoT nodes (e.g., interaction with GPIO).
- You may to replace this app with another one, of similar structure and complexity.
  - as long as it is based on sensing
- You'll need to prepare environment files.

APPLICATION 2 (2/2)

- The sensing process:
  - checks a selected GPIO pin (configured as input)
  - detects changes of the state of the pin
  - when a change (0->1 or 1->0) is detected, it puts a ”change tuple” with info on the ”direction” of the change

- The counting process:
  - inputs change tuples (from any sensing process)
  - counts the number of changes 0->1 reported by all sensing processes
  - counts the number of changes 1->0 reported by all sensing processes
  - periodically prints the two counts

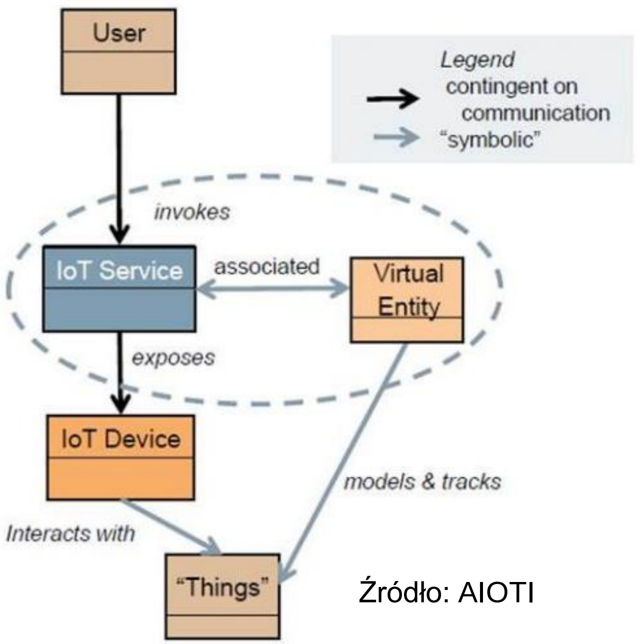
- At least two instances of the sensing process should be created.

Can App 1 and App 2 run at the same time?

- If you add to every tuple a field containing a unique application ID, the applications will not interfere with one another.
  - this should be the first field of every tuple
  - even if for some reason all the other fields of tuples used by different applications are the same, each application will handle only its own tuples

- In general, if a scheme to assign the unique application ID is agreed upon, individual applications can be developed separately.

Your results



Źródło: AIOTI

YOUR RESULTS

- source code
- a report
- a demonstration

YOUR RESULTS ARE YOURS!!!

- Freely talk with other students about concepts.
- You may reuse some items developed by another team, as long as you give credit to the team that created them.
  - this is a general rule that helps you avoid plagiarism
  - however, we reserve the right to make a judgment as to how important the reused items are, and to <u>deduct points accordingly</u>
  - reusing somebody else’s work may turn out very costly in terms of points deducted

- We can easily see reusability!
- If we see reused items without clearly given credit, all teams with those items (<u>including the authors</u>) will have the same number of points deducted.
  - (we cannot investigate who has taken what from whom)

- Your best strategy, if you are cooperative: <u>offer others advice but do not share items you are going to submit as results</u>!

THE CONTENTS OF YOUR REPORT

1. a specification of your ALP
   - produce message formats and binary encodings for the protocol messages
   - when specifying message formats, follow a good example, e.g., RTP or IP
2. a description of your tuple space API implementation (on IoT nodes)
3. a description of your server implementation
   - components (overview of the server architecture)
   - major data structures (e.g., the tuple space data structure)
4. a description of your App 1 functionality and implementation
5. a description of your App 2 functionality and implementation
6. a description of environment files for App 2

YOUR DEMO

- prepare several nodes for App 1 and App 2
  - each node for App 2 may require its own environment file
- during the demo
  - start your server with the tuple space
  - demonstrate App 1
    - comment on diagnostic messages from the broker
    - comment on what the application processes are printing
  - demonstrate App 2
    - comment on diagnostic messages from the broker
    - comment on what the application processes are printing

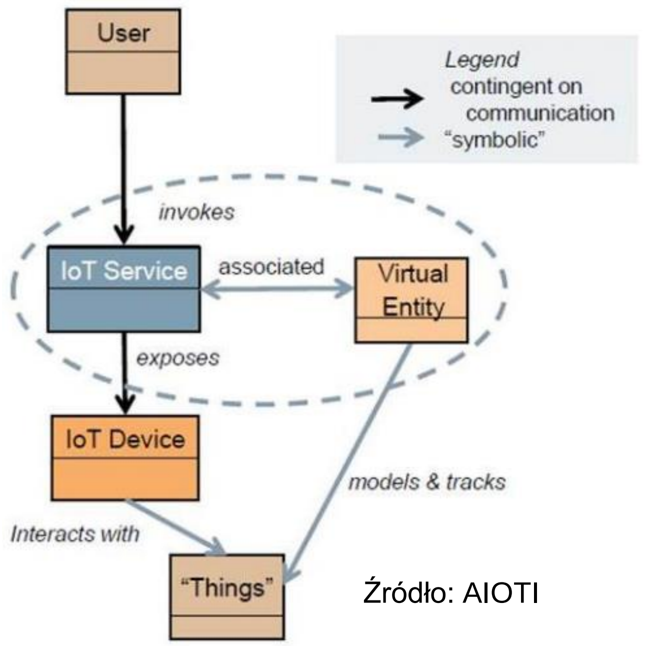
GRADING

- source code quality 5%
- report 55%
  - 1-20%, 2-10%, 3-10%, 4-5%, 5-5%, 6-5%
  - (see the contents of the report)
- demo 40%
  - what counts most is whether your software works

Deadline

- All teams must upload their results by <u>January 15, 2024 12:00.</u>
  - that's Monday
- Uploading to the PSIR-supplied git account (also used for labs).

References



Źródło: AIOTI

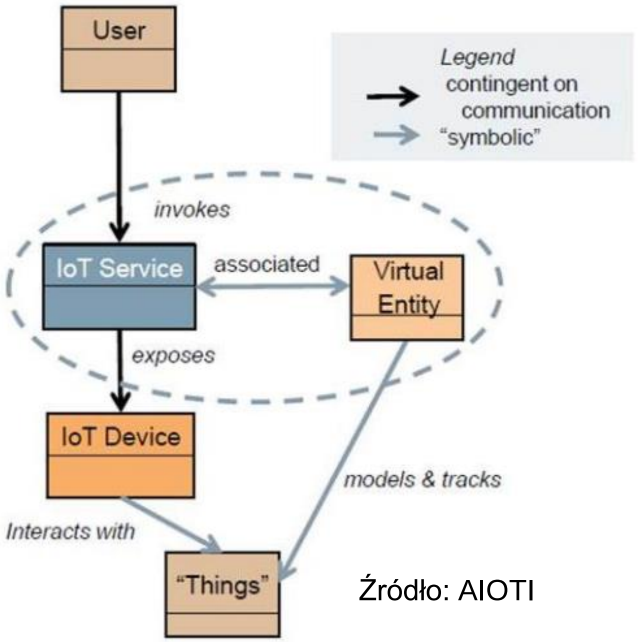
References

S. Ahuja, N. Carriero, D. Gelernter. *Linda and Friends.* Computer 19, 8 (August 1986), 26–34. https://doi.org/10.1109/MC.1986.1663305

N. Carriero, D. Gelernter*. Linda in context.* Commun. ACM 32, 4 (April 1989), 444–458. https://doi.org/10.1145/63334.63337

Vitaly Buravlev, Rocco De Nicola, Claudio Antares Mezzina. *Tuple Spaces Implementations and Their Efficiency.* 18th International Conference on Coordination Languages and Models (COORDINATION), 2016, pp.51-66. https://doi.org/10.1007/978-3-319-39519-7_4

Dziękujemy za uwagę!



Źródło: AIOTI