# Tworzenie i wykonywanie aplikacji naukowych w paradygmacie przepływu pracy

**Bartosz Baliś**

Katedra Informatyki
Wydział Informatyki, Elektroniki i Telekomunikacji AGH

Seminarium Katedry Inżynierii Oprogramowania, Politechnika Wrocławska, 21.11.2018

# Zainteresowania naukowe

- Automatyzacja obliczeń naukowych
- Naukowy workflow (scientific workflows)
- Rozproszone infrastruktury obliczeniowe (gridy, chmury)
- Obliczenia pilne (urgent computing)
  - Systemy monitorowania inteligentnych wałów powodziowych
  - Systemy wczesnego ostrzegania przed katastrofami naturalnymi

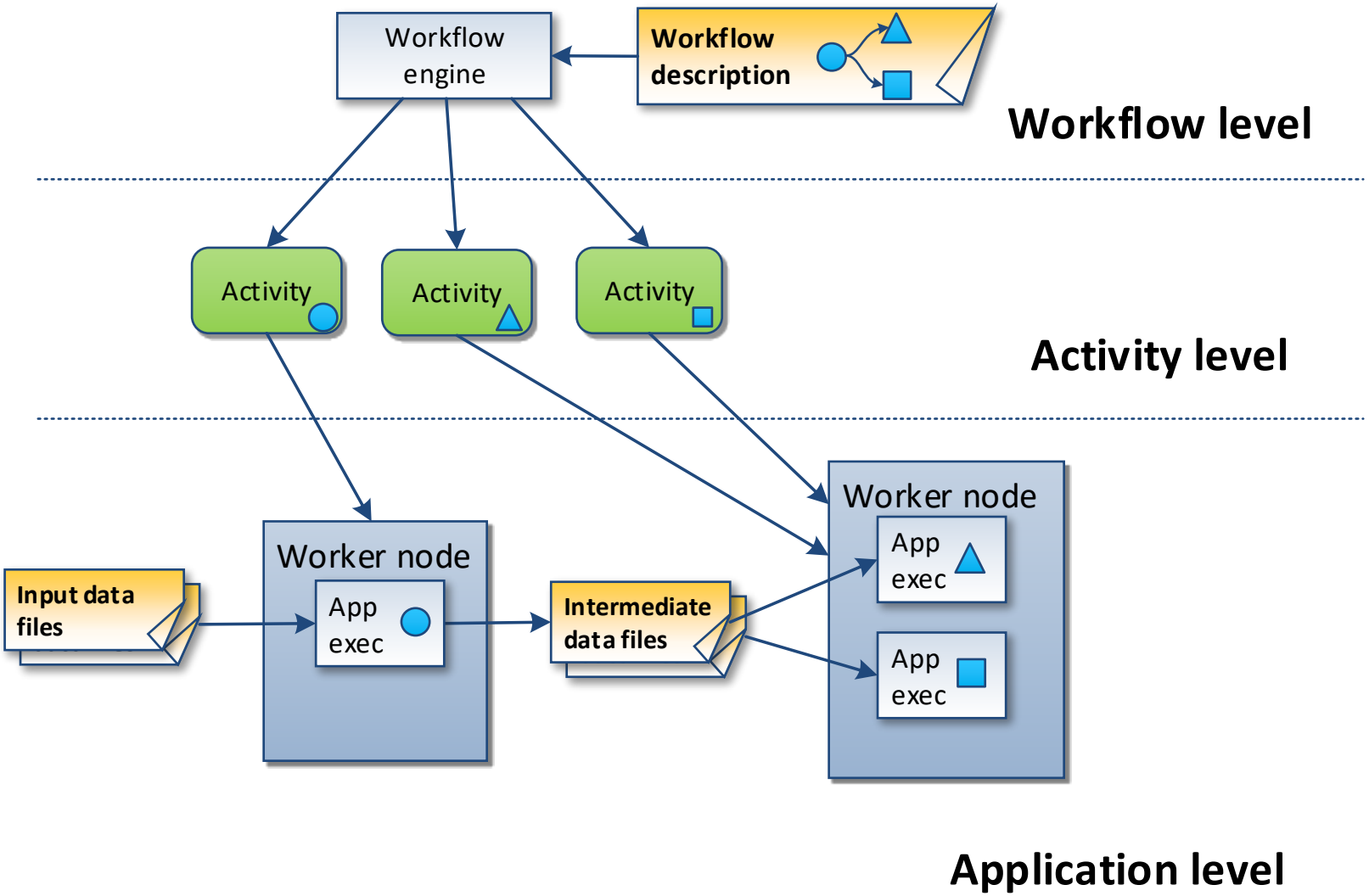
# Motywacja

- Naukowy workflow: podstawowy paradygmat automatyzacji nauki
  - Mało znany 10 lat temu, tysiące użytkowników dzisiaj
  - Szeroko wykorzystywany w naukach obliczeniowych
- Rozwój nauk obliczeniowych: extreme data, extreme computing, extreme complexity
  - Dokładniejsze modele, zalew danych
  - Połączenie symulacji z przetwarzaniem Big Data, machine learning
- Rozwój infrastruktur obliczeniowych: nowe architektury, modele dostępu do zasobów
  - chmury IaaS, PaaS, bezserwerowe; gridy; systemy HPC
- Rosnąca baza i różnorodność użytkowników
  - Domain scientist, research developer, data scientist, system engineer

→ **Ciągłe wyzwania dla automatyzacji obliczeń naukowych**

→ **Potrzeba produktywnych środowisk dla naukowych przepływów**

# Naukowy workflow: anatomia



# System zarządzania przepływem naukowym

Workflow Management System (WfMS)

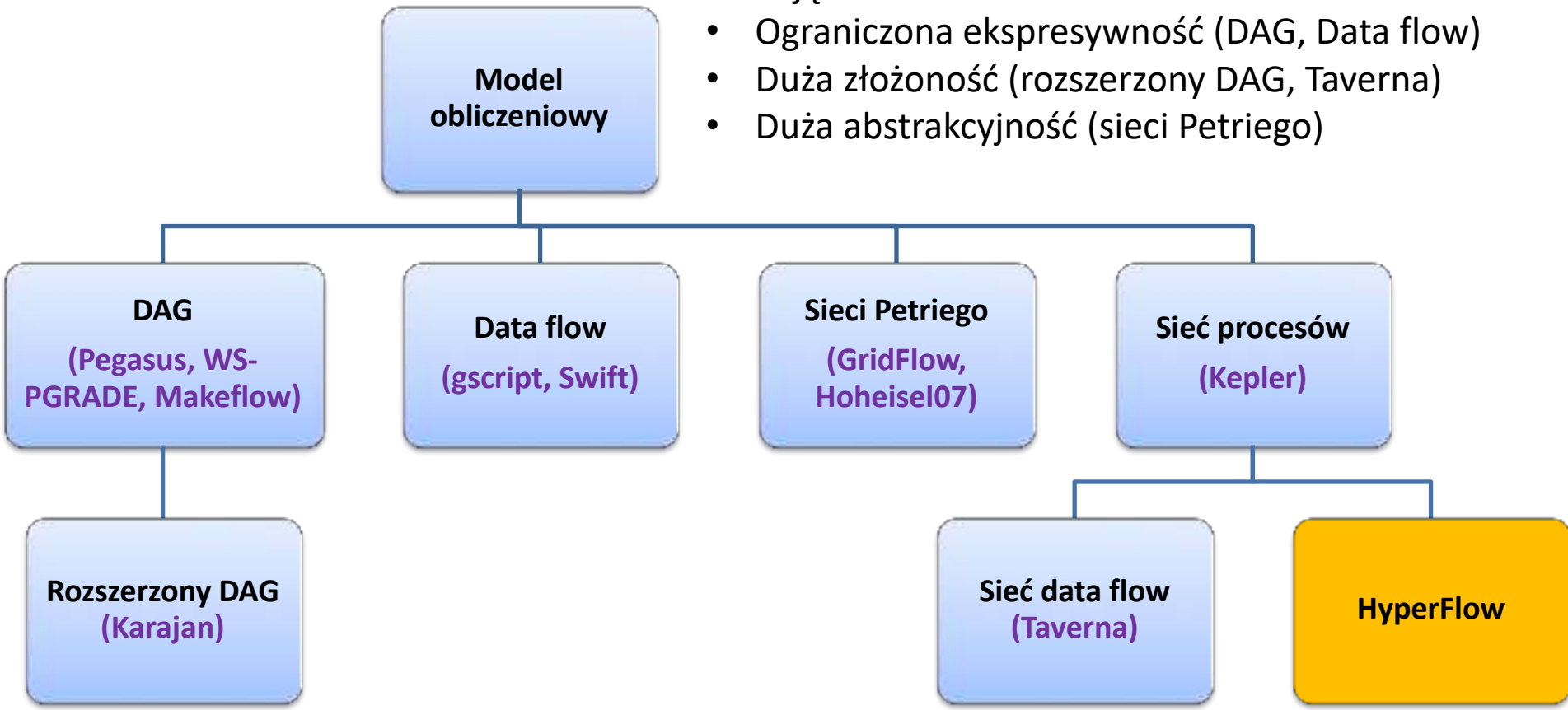
- Narzędzia (język, API) do programowania naukowych workflow
- Środowisko wykonawcze dla workflow
  - Obliczenia w rozproszonych infrastrukturach obliczeniowych
  - Planowanie / optymalizacja wykonania (scheduling)
  - Zarządzanie stanem (utrwalanie i odtwarzanie)
  - Obsługa awarii
  - Logowanie danych provenance
- Istniejące systemy: Pegasus, Taverna, Kepler, Triana, Makeflow, WS-PGrade, Swift, **HyperFlow**


# MODEL OBLICZENIOWY I PROGRAMOWANIE

# Stan wiedzy (1): model obliczeniowy

Istniejące modele:

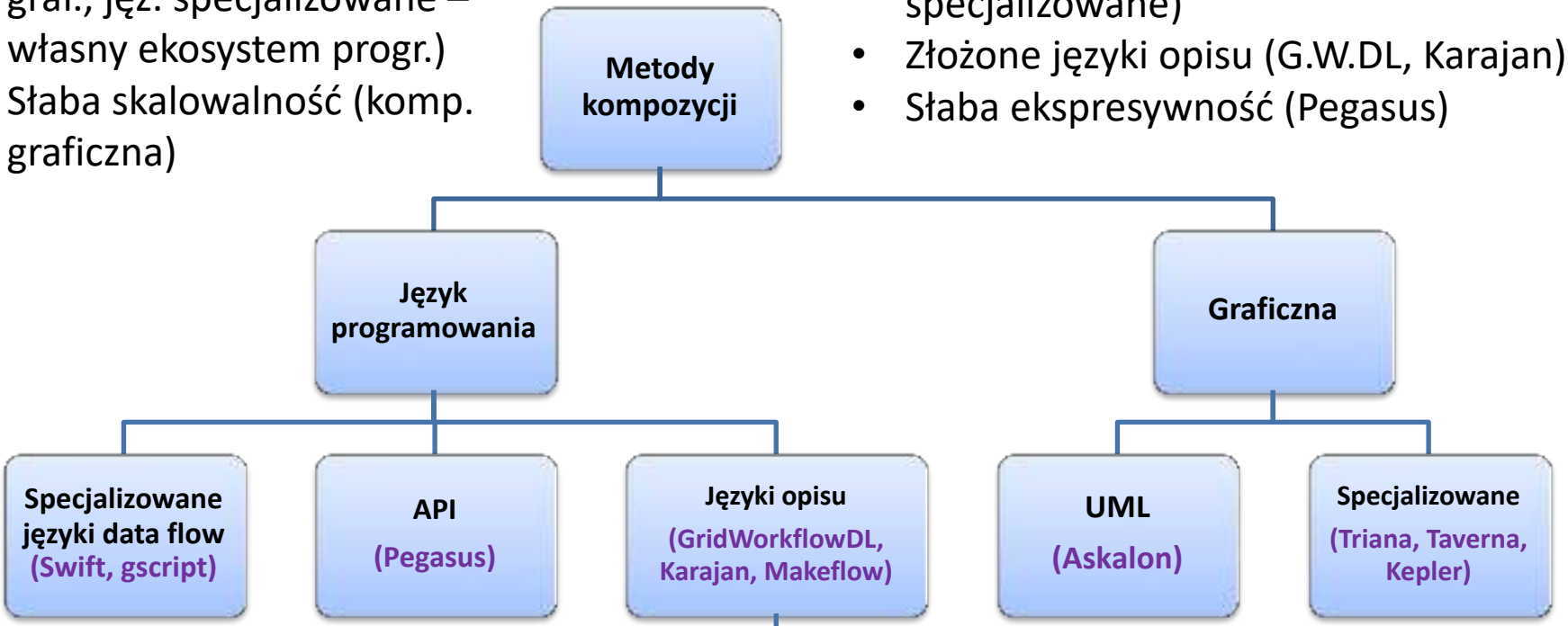
- Ograniczona ekspresywność (DAG, Data flow)
- Duża złożoność (rozszerzony DAG, Taverna)
- Duża abstrakcyjność (sieci Petriego)



# Stan wiedzy (2): programowanie – kompozycja grafu

Ograniczenia istniejących metod:

- Niska produktywność (komp. graf., jęz. specjalizowane – własny ekosystem progr.)
- Słaba skalowalność (komp. graficzna)
- Trudność generowania workflow z szablonu (komp. graf., jęz. specjalizowane)
- Złożone języki opisu (G.W.DL, Karajan)
- Słaba ekspresywność (Pegasus)



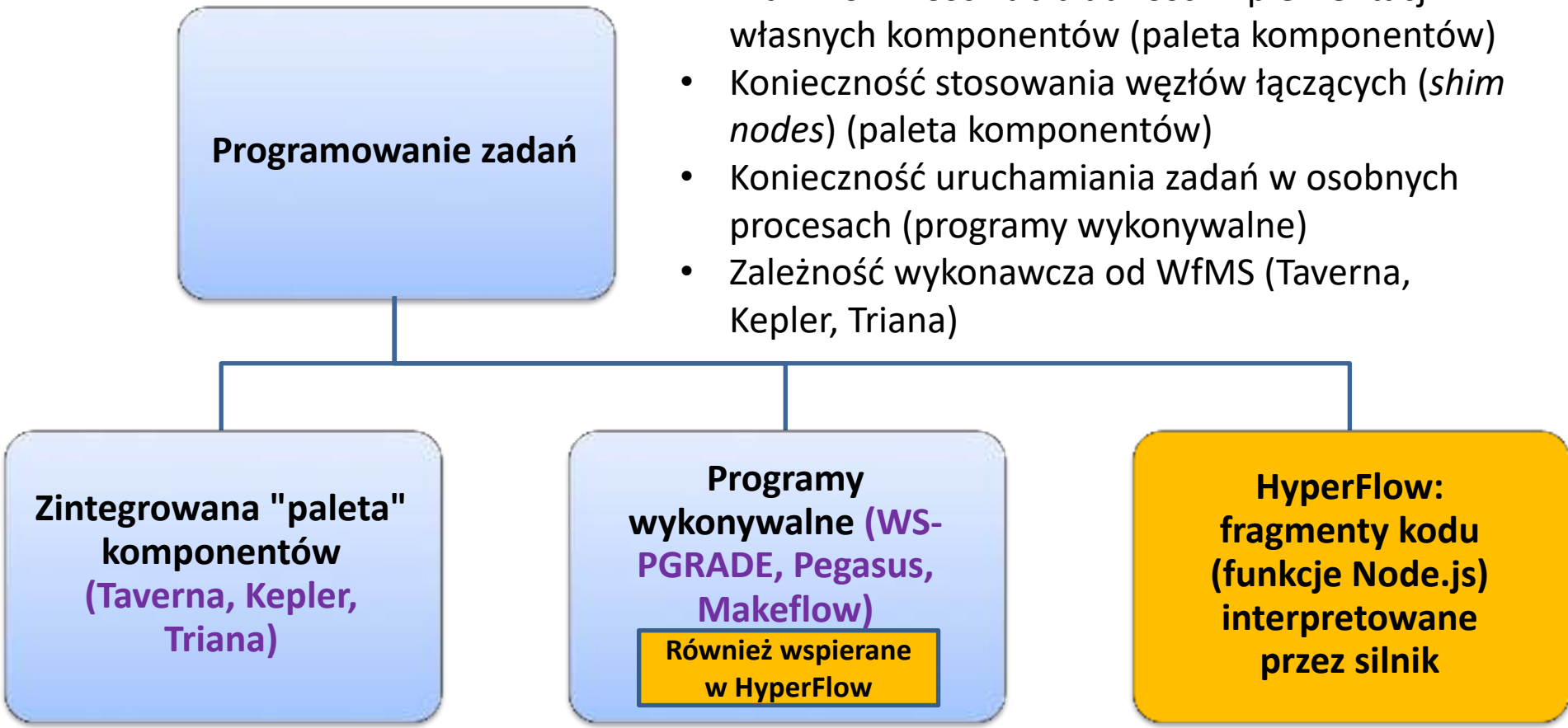

- Prosty i ekspresywny język opisu
- Łatwość generowania dużych grafów (tysiące węzłów)

**HyperFlow**

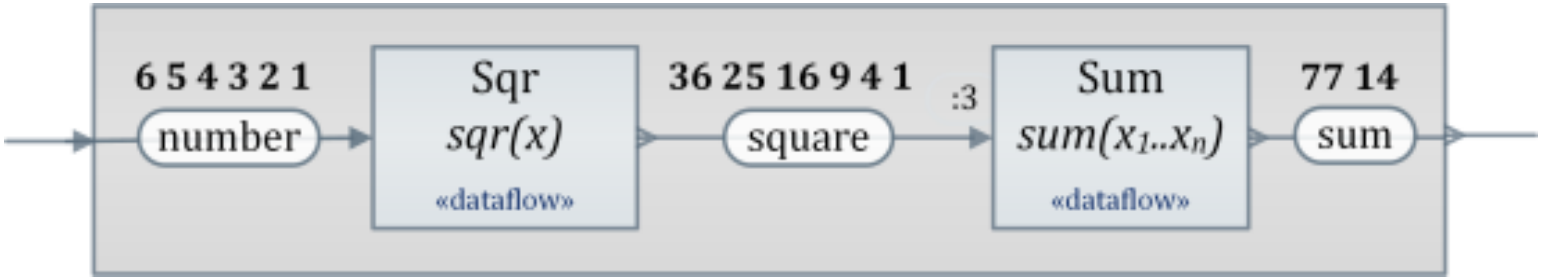
# Stan wiedzy (3): Programowanie zadań

Ograniczenia istniejących metod:

- Brak możliwości lub trudność implementacji własnych komponentów (paleta komponentów)
- Konieczność stosowania węzłów łączących (*shim nodes*) (paleta komponentów)
- Konieczność uruchamiania zadań w osobnych procesach (programy wykonywalne)
- Zależność wykonawcza od WfMS (Taverna, Kepler, Triana)



# Programowanie w HyperFlow




<workflow.json>

```
{
  "name": "Wf_SumSquares",
  "processes": [ {
    "name": "Sqr",
    "type": "dataflow",
    "function": "sqr",
    "parlevel": 0,
    "ordering": "true",
    "ins": [ "number" ],
    "outs": [ "square" ]
  }, {
    "name": "Sum",
    "type": "dataflow",
    "function": "sum",
    "ins": [ "square:3" ],
    "outs": [ "sum" ]
  } ],
  "signals": [ {
      "name": "number",
      "data": [ 1, 2, 3, 4, 5, 6 ]
    }, {
      "name": "square"
    }, {
      "name": "sum"
    } ],
  "ins": [ "number" ],
  "outs": [ "sum" ]
}
```

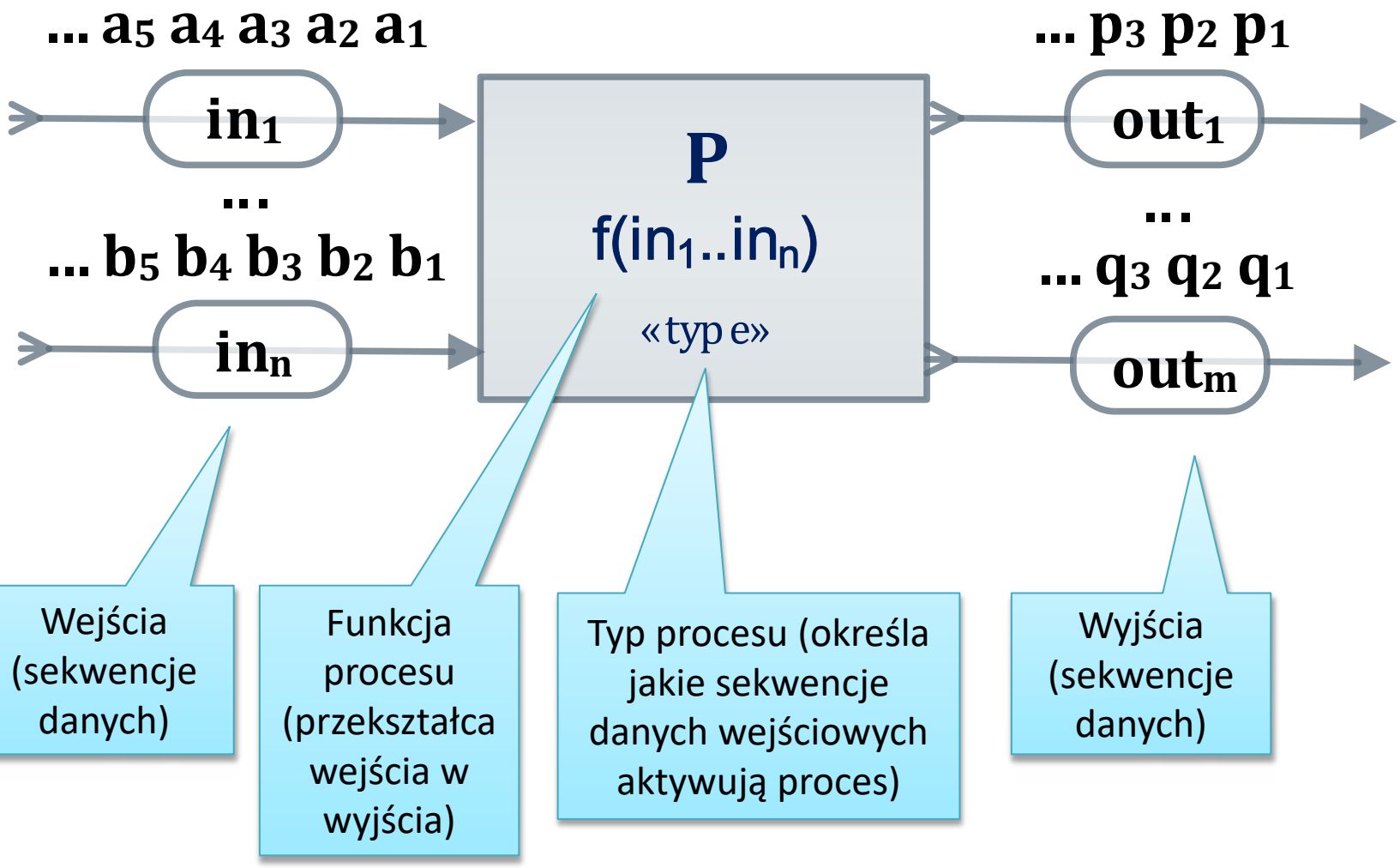
Przetwarza dane wejściowe równolegle, zachowuje kolejność wyników

<functions.js>

```
function sqr(ins, outs, config, cb) {
  var n = Number(ins.number.data[0]);
  outs.square.data = [n * n];
  cb(null, outs);
}

function sum(ins, outs, config, cb) {
  var sum=0.0;
  ins.square.data.forEach(function (n) {
    sum += n;
  });
  outs[0].data = [ sum ];
  cb(null, outs);
}
```

# Proces HyperFlow



# Funkcja procesu

Read inputs

Write outputs

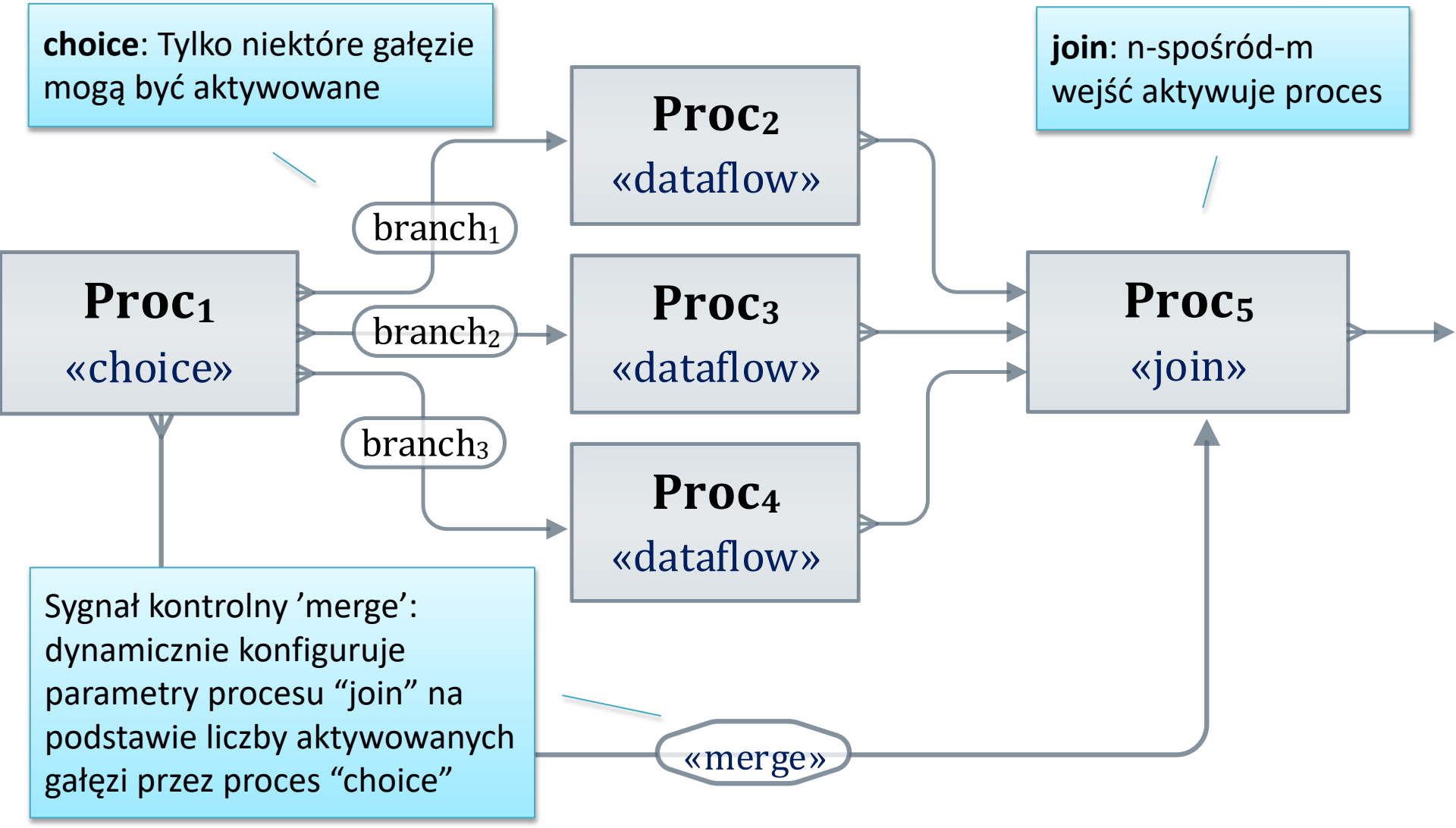
Return outputs

```
function sqr(ins, outs, context, cb) {
    var n = Number(ins.number.data[0]);
    outs.square.data = [n * n];
    cb(null, outs);
}
```

Aktualnie wspierany runtime: JavaScript/Node.js

Samodzielna funkcja, nie ma zależności wykonawczej od HyperFlow

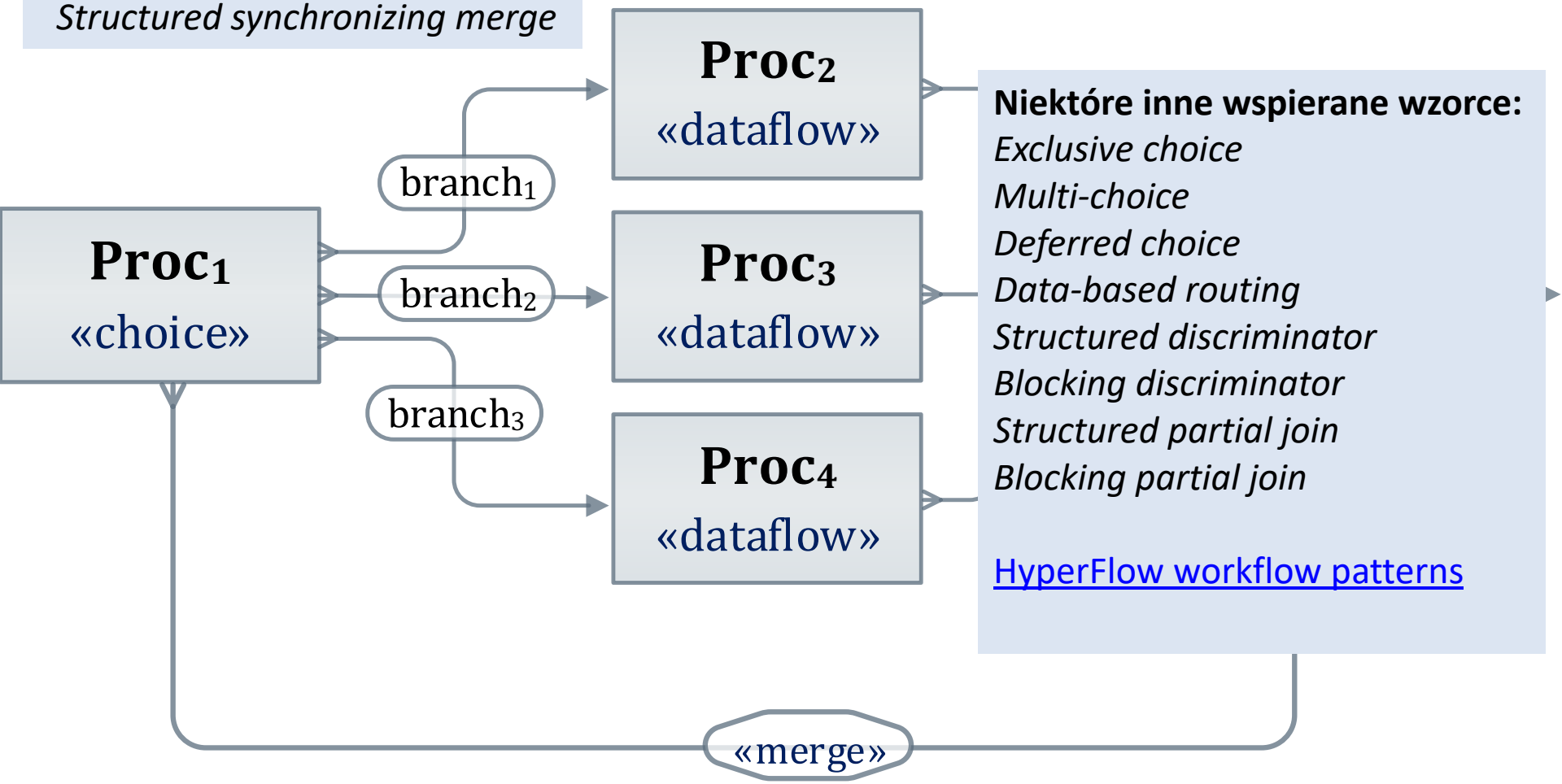
# Ekspresywność języka: złożone wzorce przepływu kontroli



# Ekspresywność języka: złożone wzorce przepływu kontroli

**Ten wzorzec:**
*Structured synchronizing merge*

www.workflowpatterns.org

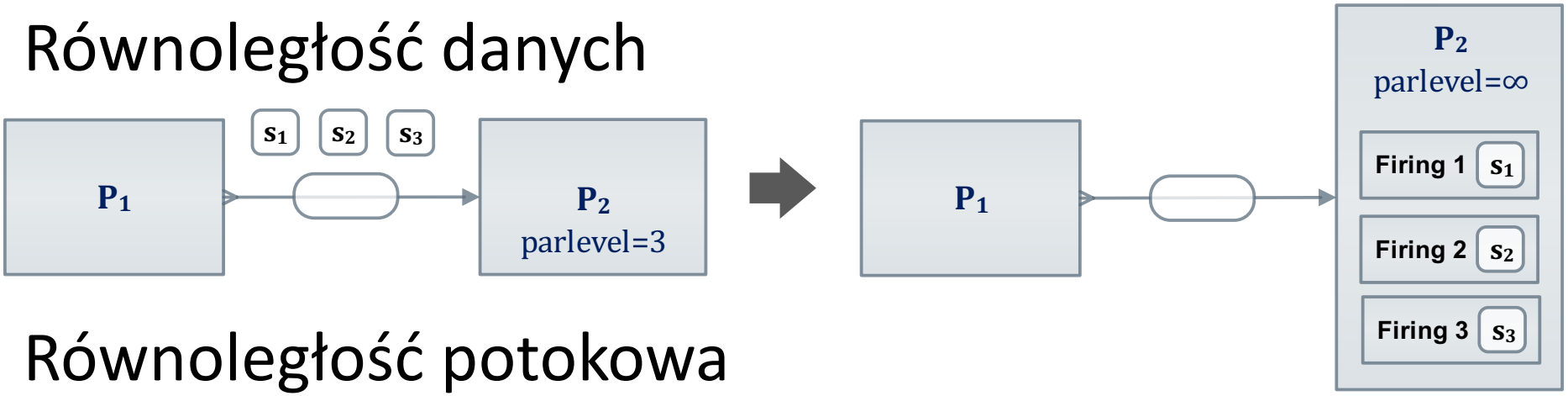



**Niektóre inne wspierane wzorce:**
*Exclusive choice*
*Multi-choice*
*Deferred choice*
*Data-based routing*
*Structured discriminator*
*Blocking discriminator*
*Structured partial join*
*Blocking partial join*

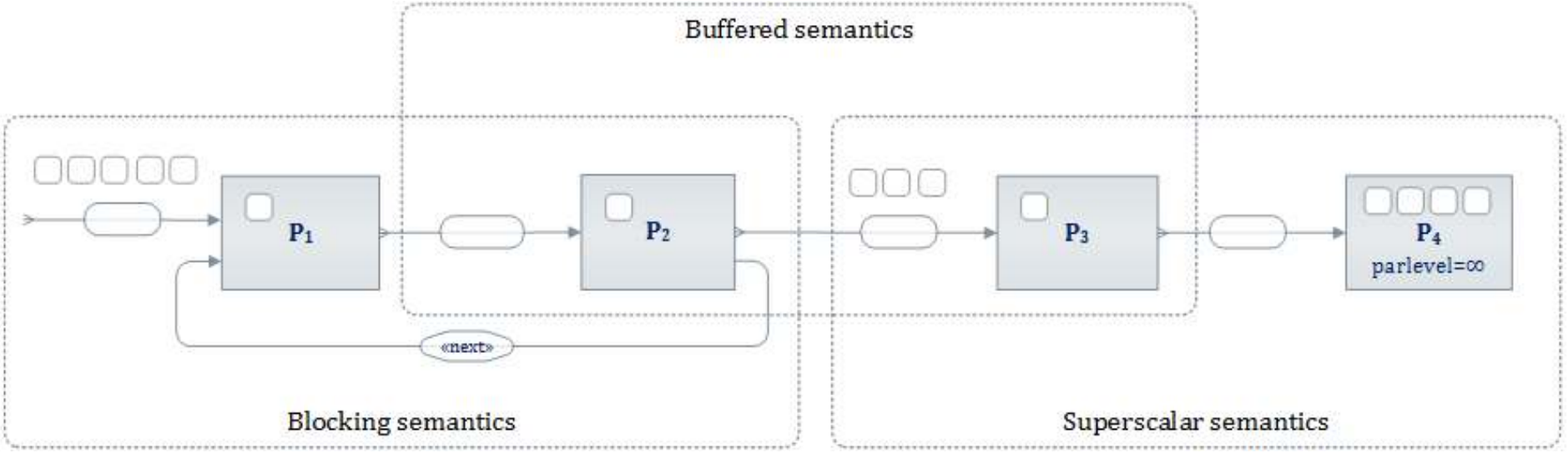
HyperFlow workflow patterns

# Ekspresywność języka: wzorce przetwarzania równoległego

## Równoległość danych




## Równoległość potokowa



# Produktywność: prostota implementacji

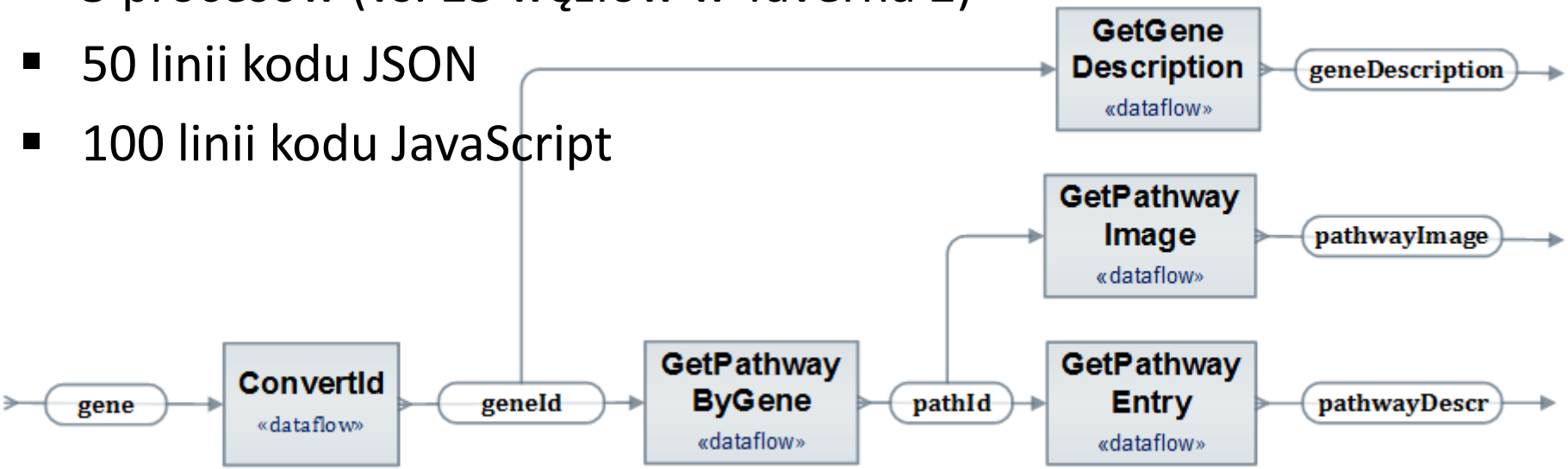
Przykład: workflow pobierający informacje o ścieżkach metabolicznych związanych z wybranymi genami

- Wywołuje 5 publicznych usług REST

Oryginalna implementacja: [myexperiment.org](myexperiment.org) (Taverna 2)

Implementacja w HyperFlow:

- **5** procesów (vs. **13** węzłów w Taverna 2)
- 50 linii kodu JSON
- 100 linii kodu JavaScript



# Model obliczeniowy i programowanie: podsumowanie

- Opracowanie modelu obliczeniowego dla przepływów naukowych **HyperFlow [H1,H2]**
- Opracowanie metody programowania przepływów naukowych **[H1,H10]**
- Implementacja systemu HyperFlow (licencja MIT Open Source) (https://github.com/hyperflow-wms/hyperflow)

**Znaczenie wyników:**

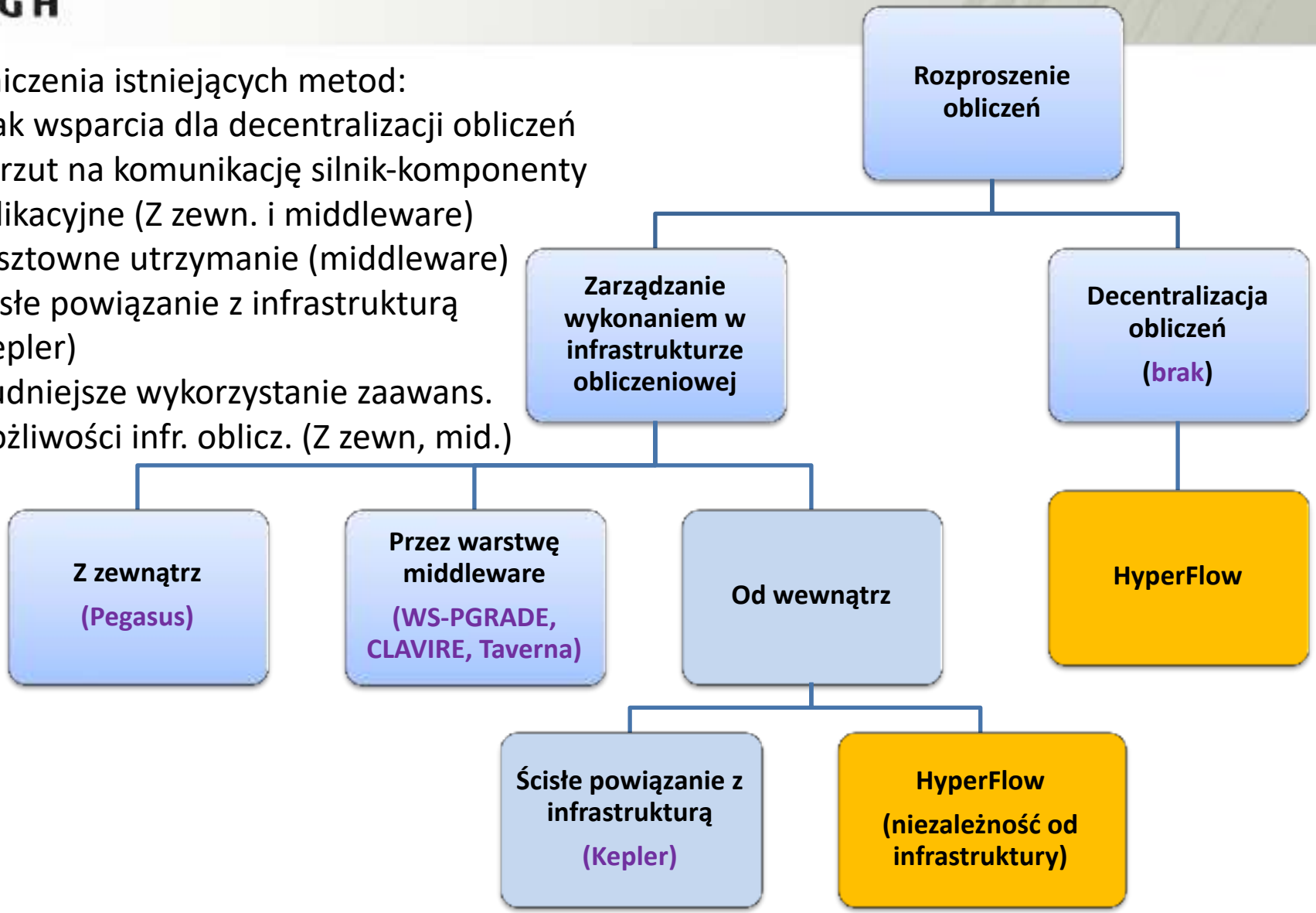
- Wysoka produktywność programistyczna (połączenie modelu data flow z programowaniem imperatywnym)
- Uproszczona implementacja przepływów (eliminacja węzłów łączących)
- Wysoka ekspresywność: łatwość wyrażania złożonych wzorców przetwarzania (przepływu kontroli i danych, przetwarzania kolekcji danych i przetwarzania równoległego)

# ROZPROSZENIE OBLICZEŃ

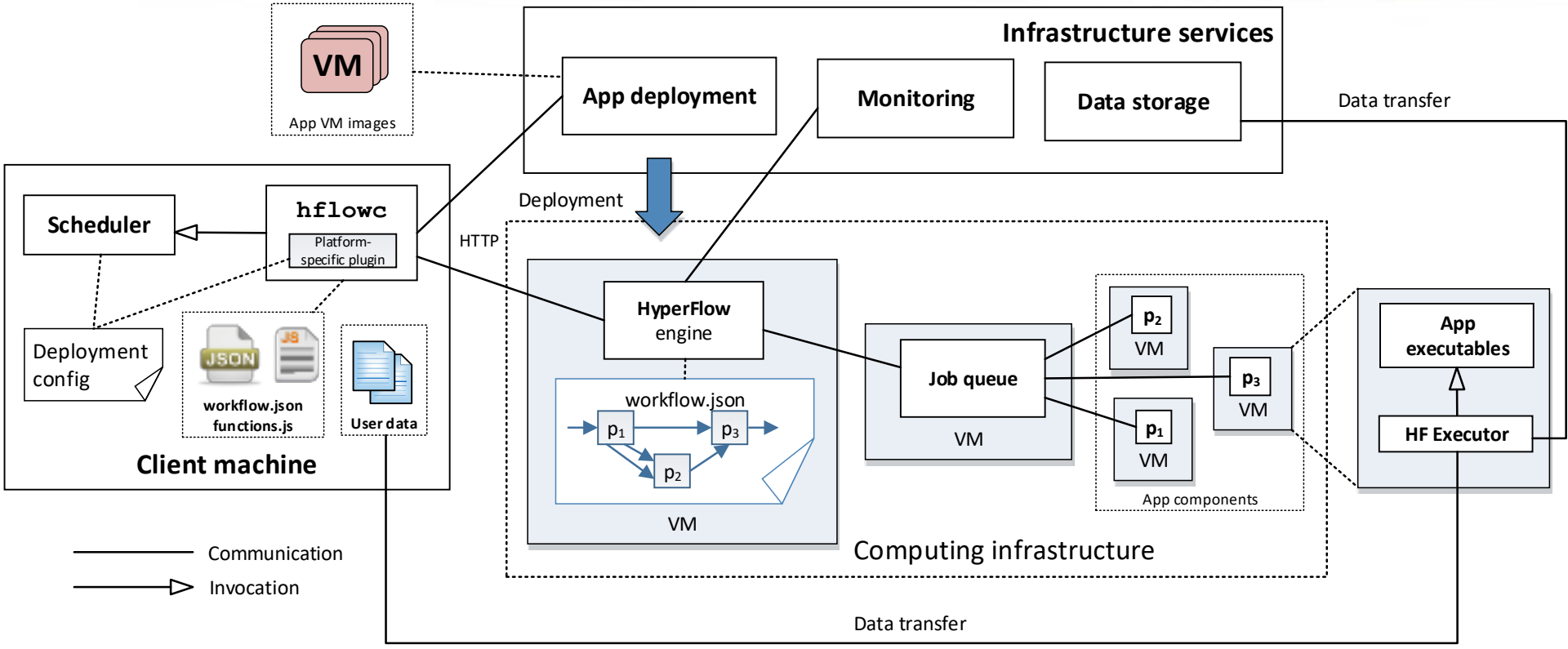
# Stan wiedzy

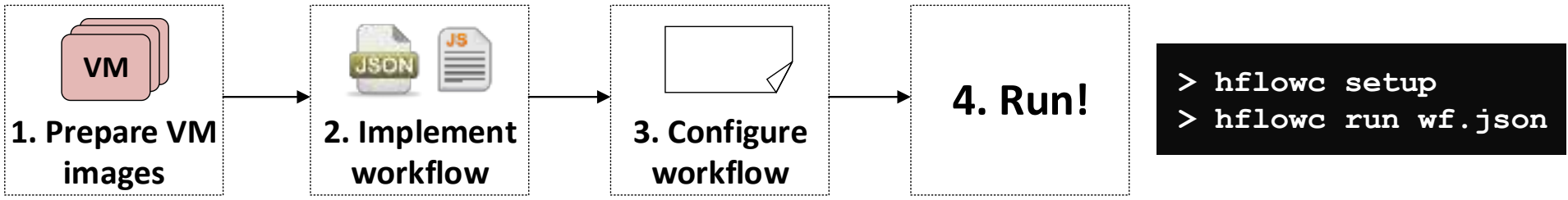
Ograniczenia istniejących metod:

- Brak wsparcia dla decentralizacji obliczeń
- Narzut na komunikację silnik-komponenty aplikacyjne (Z zewn. i middleware)
- Kosztowne utrzymanie (middleware)
- Ścisłe powiązanie z infrastrukturą (Kepler)
- Trudniejsze wykorzystanie zaawans. możliwości infr. oblicz. (Z zewn, mid.)

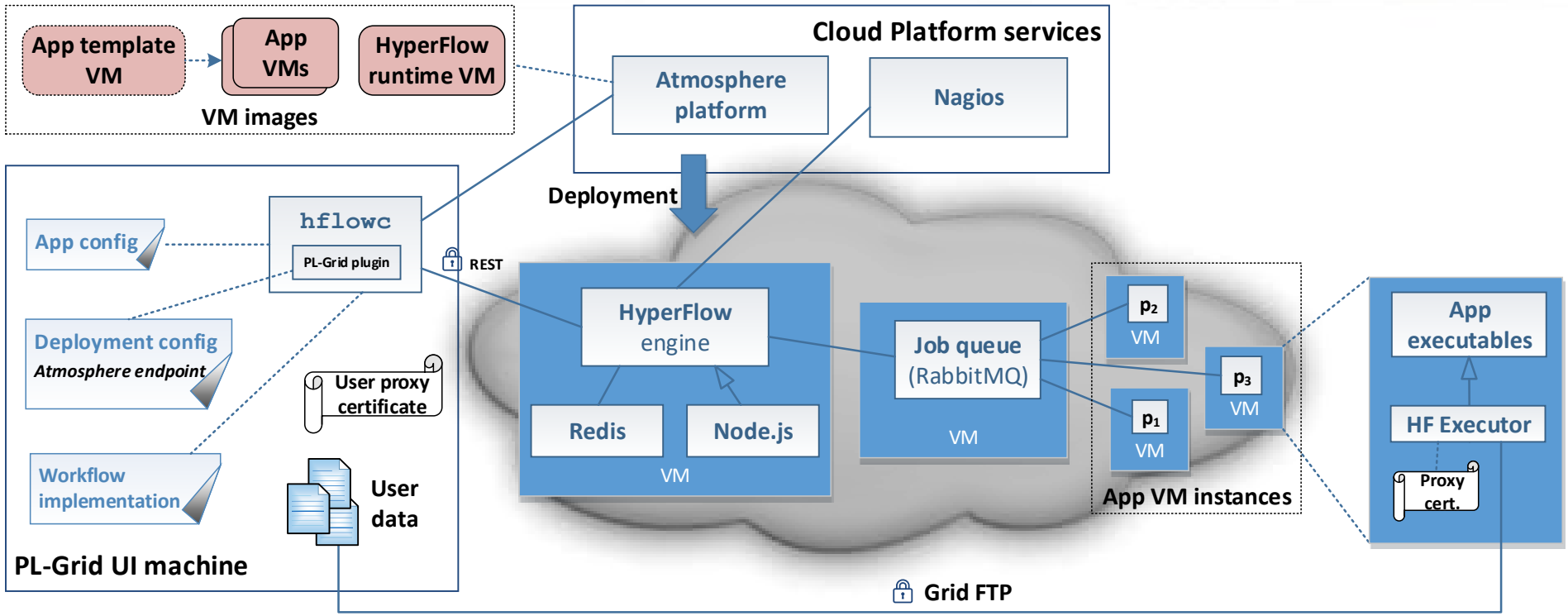


# Rozproszona architektura wykonawcza HyperFlow



# Konkretna realizacja (1): PL-Grid




- Oficjalne wdrożenie jako usługa w infrastrukturze PL-Grid

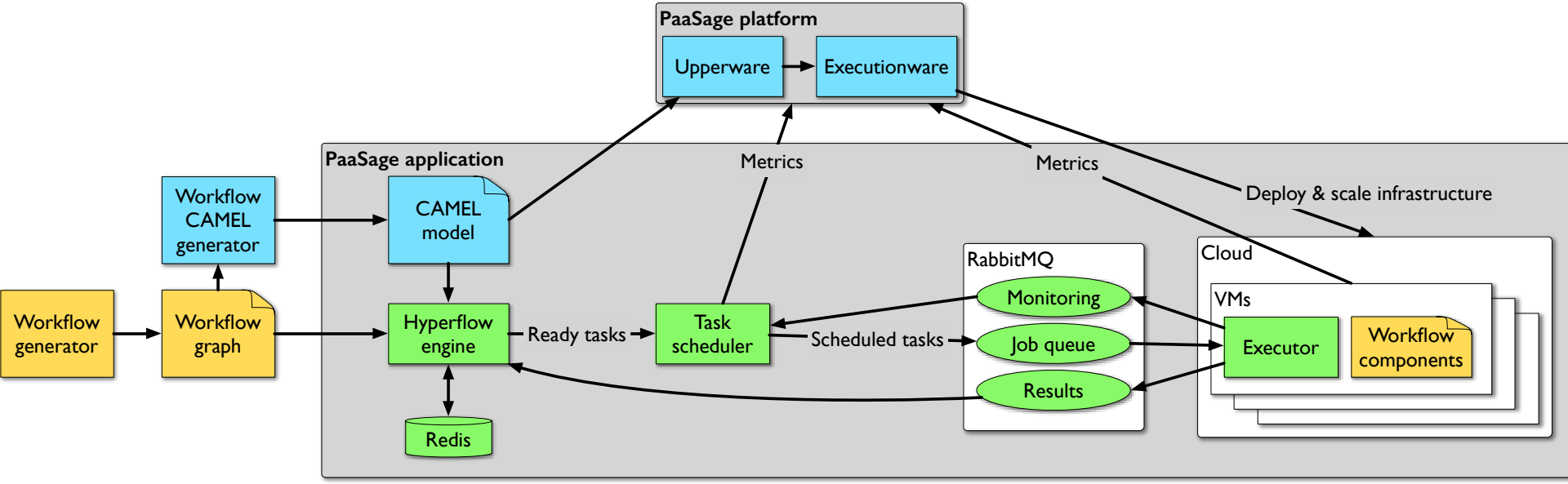
# Konkretna realizacja (2): platforma PaaSage

**PaaSage cloud platform**:

- Model-based development of cloud applications
- Multi-cloud application deployment
- Autoscaling according to application demands

**Integration of HyperFlow with PaaSage:**

- CAMEL app model generated from HyperFlow Wf description
- Task scheduler delivers **initial deployment plan** and **scalability rules**
- **Workflow monitoring** information triggers workflow autoscaling



# Korzyści

Lepsza wydajność komunikacji silnik – węzły obliczeniowe

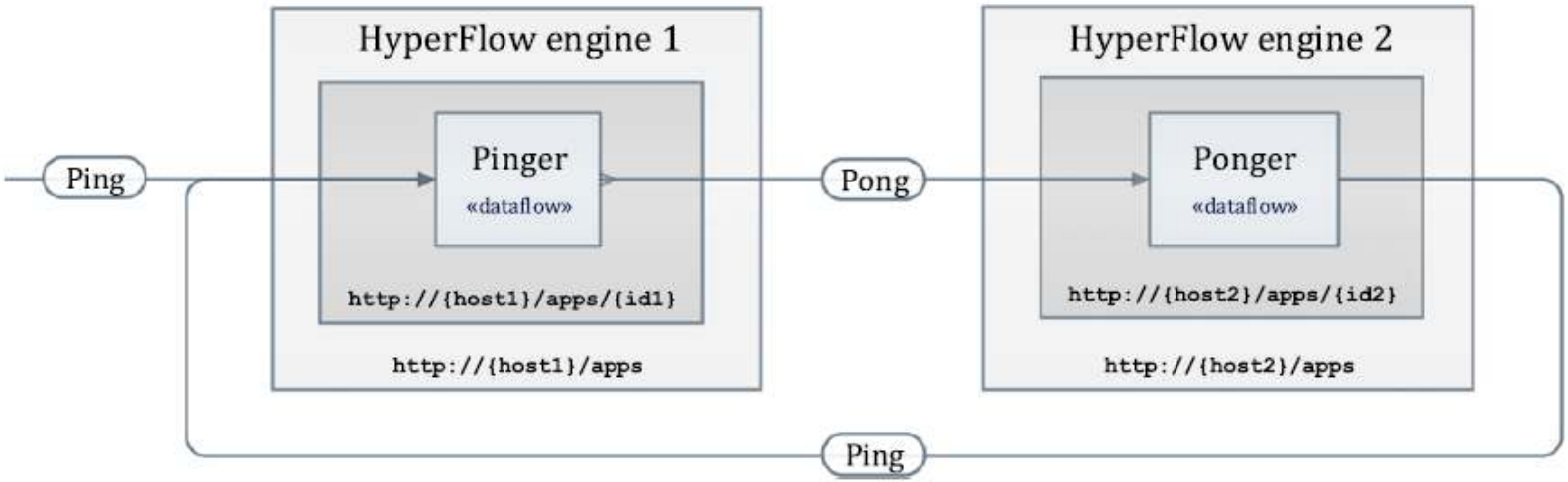
| Variant | Execution time (4x6vCPU) | Execution time (1x6vCPU) |
|---|---|---|
| Local wf engine | $5\,min 48\,s \pm 3\,s$ | $25\,min 7\,s$ |
| Remote wf engine | $8\,min 43\,s \pm 2\,s$ (150 %) | $26\,min 2\,s$ (104 %) |

Lepsza integracja z usługami danej platformy chmurowej

- Przykład zastosowania: automatyczne skalowanie

Łatwiejsze utrzymanie

# Decentralizacja: rozproszony, zdecentralizowany workflow z cyklem




| Operation & URI | Body | Effect |
|---|---|---|
| POST {appfactory1} | Pinger.json | Create Pinger instance, return URI {app1} |
| POST {appfactory2} | Ponger.json | Create Ponger instance, return URI {app2} |
| PUT {app1}/sigs/Ping/remotesinks | {"uri":{app2}} | Connect signal Ping from app1 to app2 |
| PUT {app2}/sigs/Pong/remotesinks | {"uri":{app1}} | Connect signal Pong from app2 to app1 |
| POST {app1} | Ping signal data | Send the initial signal to start Ping-Pong |

# Rozproszenie obliczeń: podsumowanie

- Zaprojektowanie protokołu rozproszonego i w pełni zdecentralizowanego wykonania przepływów **[H1,H10]**
- Opracowanie nowej metody i architektury umożliwiającej wykonywanie przepływów naukowych w rozproszonych infrastrukturach obliczeniowych **[H7],** w tym badanie infrastruktur bezserwerowych **[H6]**

**Znaczenie wyników:**

- Możliwość kompozycji niezależnych luźno powiązanych przepływów w jedno obliczenie
- Zmniejszenie narzutu na komunikację pomiędzy silnikiem a komponentami aplikacyjnymi
- Możliwość skuteczniejszego zarządzania komponentami wykonawczymi (np. skalowanie)
- Uniwersalność zaproponowanej architektury – pokazano jej zastosowanie w chmurze Amazon, chmurze PLGrid, chmurze typu PaaS i klastrze PBS

**Zastosowania**:

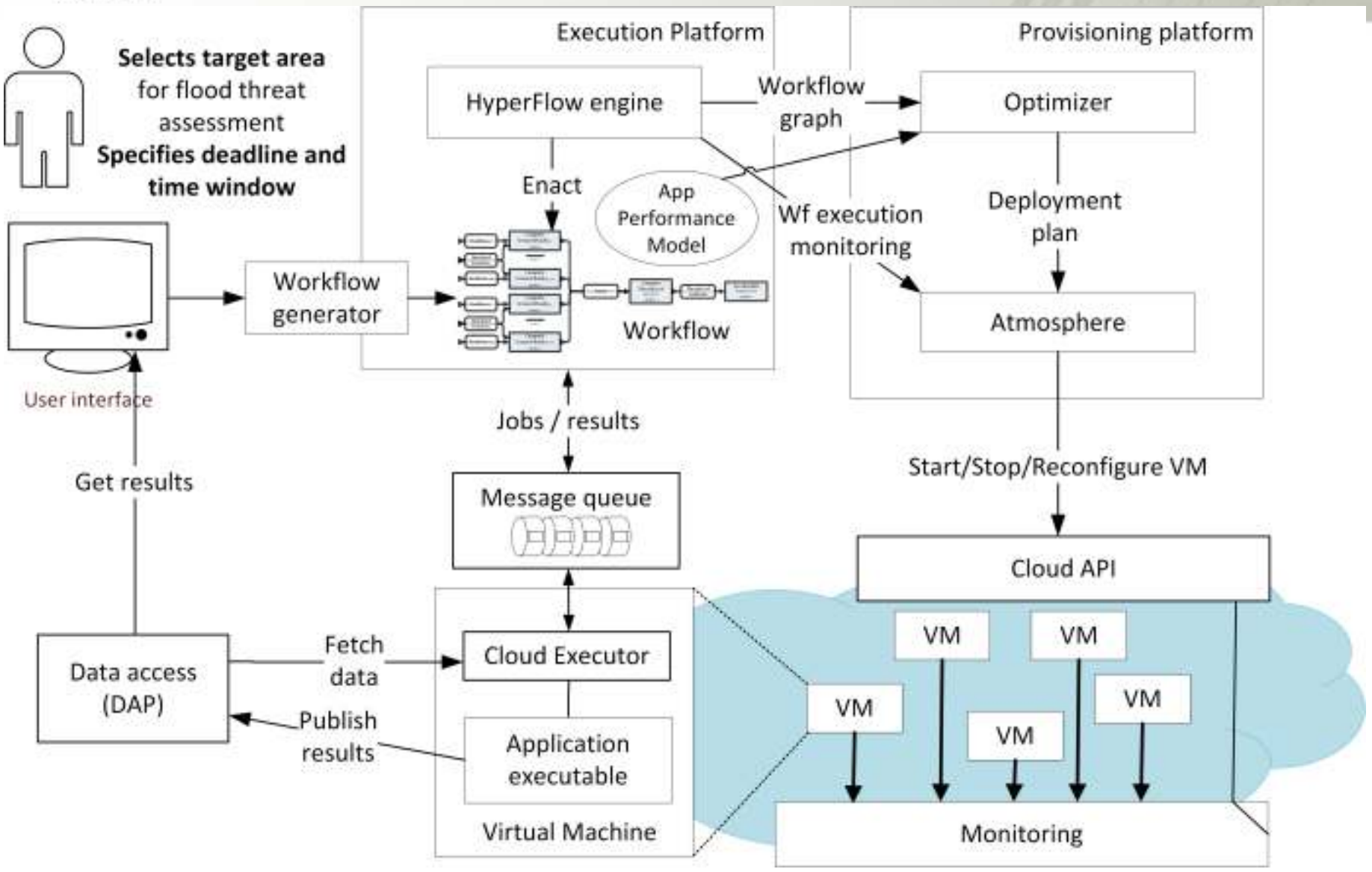
- Wdrożenie systemu HyperFlow jako usługi dla obliczeń naukowych w infrastrukturze PLGrid

# ZASTOSOWANIE PRAKTYCZNE

# Zarządzanie obliczeniami pilnymi

- Scenariusz: obliczanie poziomu zagrożenia powodziowego dla dużych obszarów obwałowań
- Obliczenia terminowe (deadline-driven)

# Zarządzanie obliczeniami pilnymi



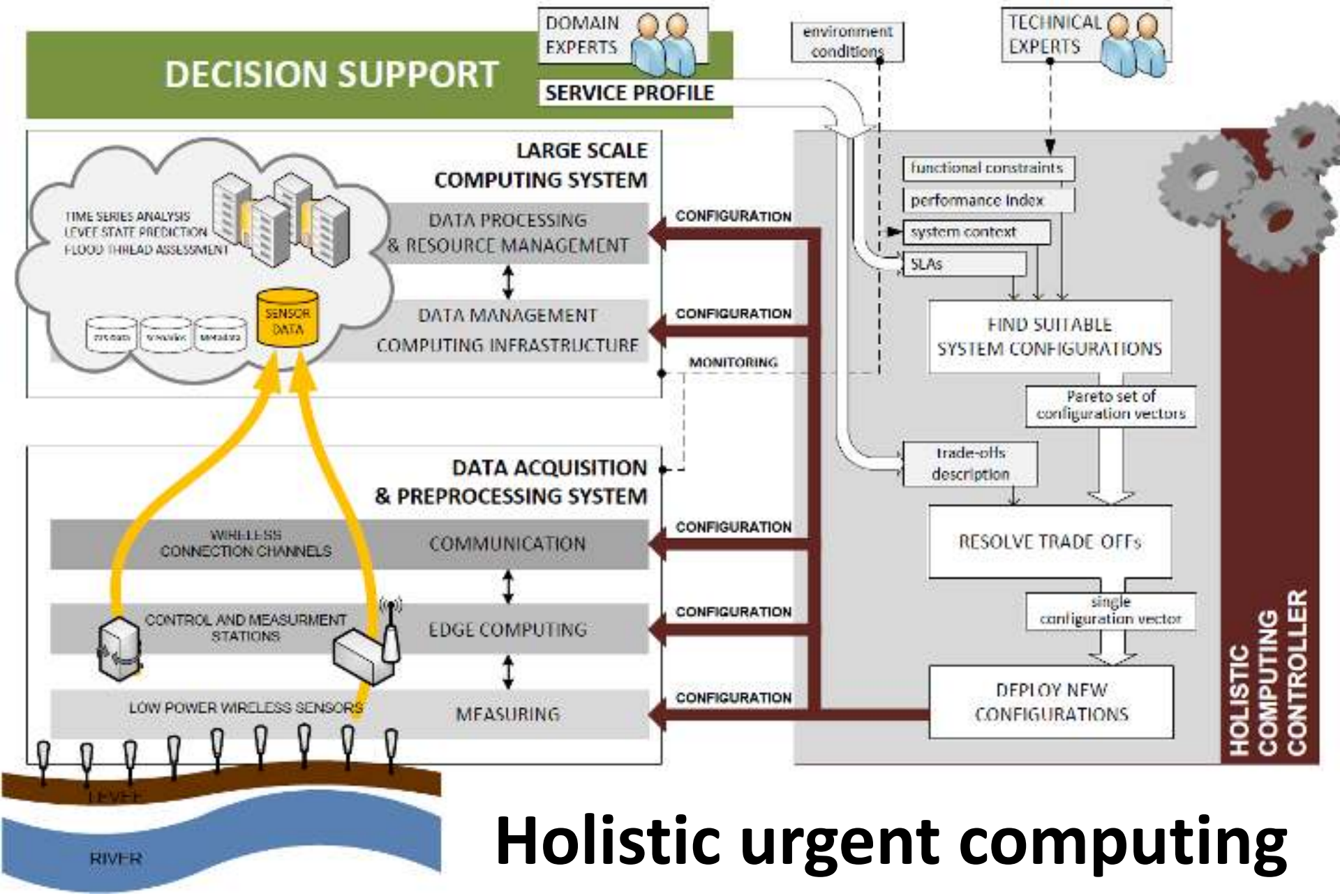
# Model wydajności

$$\boldsymbol{T} = a * \frac{\boldsymbol{s}*\boldsymbol{d}}{\boldsymbol{v}} + b * \boldsymbol{v} + c \qquad (1)$$

$\boldsymbol{T}$ – *total computing time*
$\boldsymbol{v}$ – number of VMs
$\boldsymbol{d}$ – time window in days
$\boldsymbol{s}$ – number of tasks (sections)

Parameters $a, b, c$ to be determined experimentally

Solve eq. (1) to compute $\boldsymbol{v}$ (# of VMs) given $\boldsymbol{T}$ (deadline)




# Holistic urgent computing management: algorithm

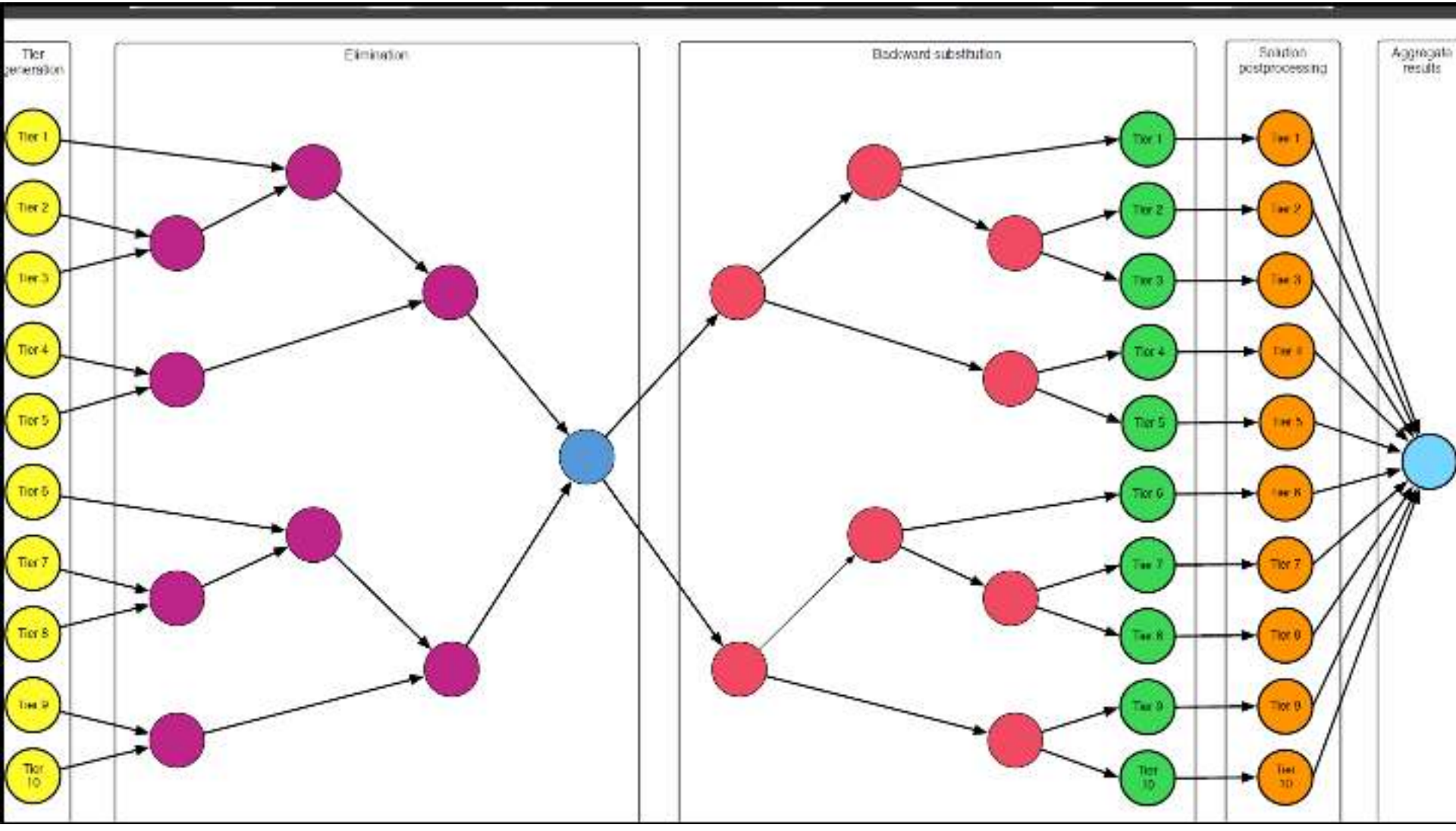
# Zastostosowanie przepływów naukowych do obliczeń pilnych

- Opracowanie metod i architektur wykonawczych dla obliczeń pilnych w oparciu o przepływ naukowy **[H9,H12]**
- Holistyczna metoda optymalizacji konfiguracji systemu do monitorowania inteligentnych wałów powodziowych **[H4]**

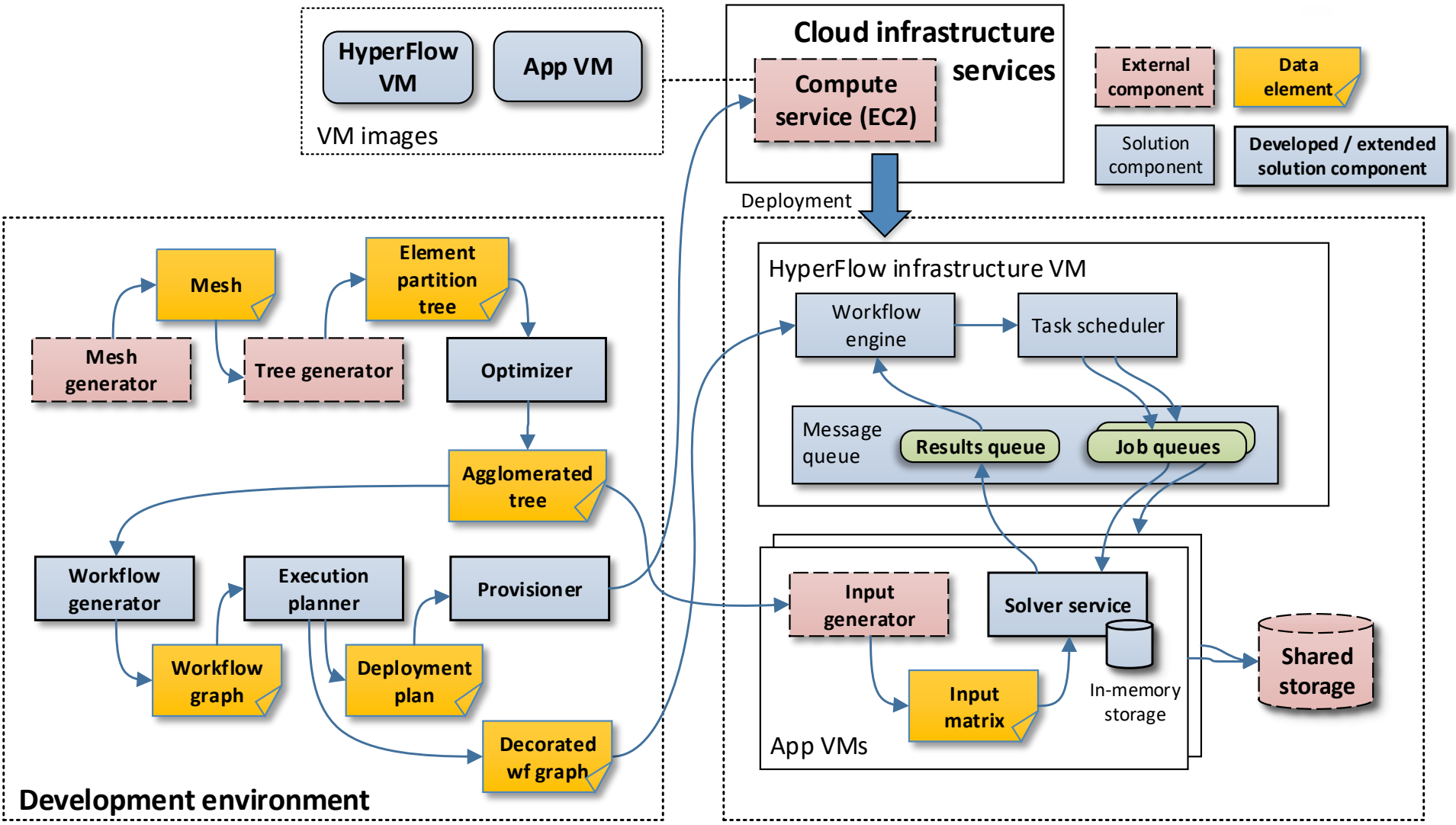
**Znaczenie wyników**:

- Pokazanie efektywności zastosowania paradygmatu przepływu naukowego do obliczeń pilnych
- Praktyczna weryfikacja HyperFlow w dziedzinie inżynierii złożonych systemów obliczeniowych

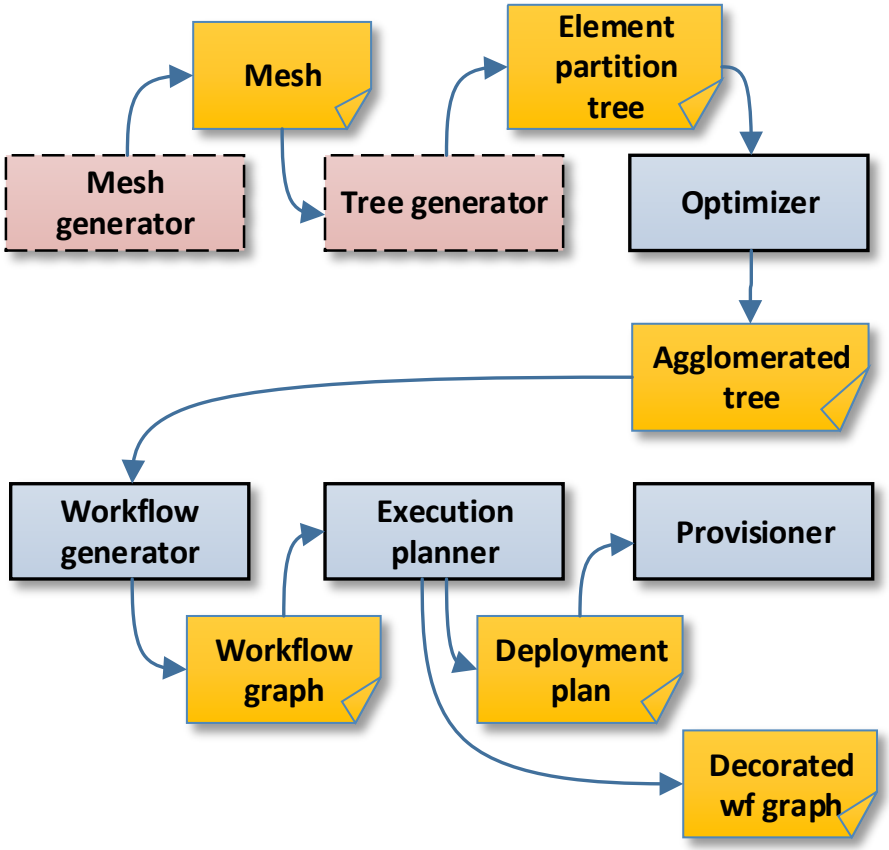
# Zrównoleglanie aplikacji HPC w chmurze (przykład solwer FEM)

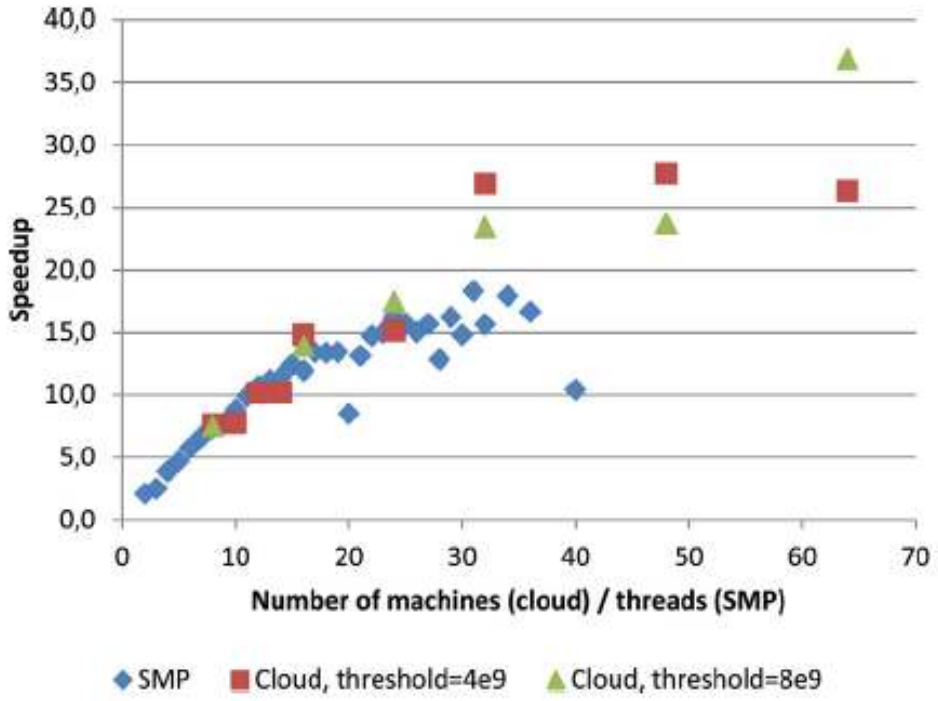


# Zrównoleglanie aplikacji HPC w chmurze: architektura



# Zrównoleglania aplikacji HPC w chmurze: wyniki



# Zespół badawczy


dice.cyfronet.pl

DiCE: Distributed Computing Environments

- Prof. Marian Bubak (szef zespołu)
- Dr hab. Maciej Malawski
- Doktoranci
  - Kamil Figiela
  - Maciej Pawlik
  - Michał Orzechowski
- Magistranci
- Współpraca
  - Prof. Ewa Deelman i jej zespół (ISI USC)
  - Prof. Maciej Paszyński i jego zespół (KI AGH)

# Bibliografia (1)


**[H1] B. Baliś**, “Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows,” Future Generation Computer Systems 55:147-162, 2016.

**[H2] B. Baliś**, K. Borowski, “Using an Actor Framework for Scientific Computing: Opportunities and Challenges,” Computing and Informatics 35(4):870–889, 2016.

**[H3] B. Baliś**, K. Figiela, K. Jopek, M. Malawski, and M. Pawlik, “Porting HPC applications to the cloud: A multi-frontal solver case study,” Journal of Computational Science 18: 106–116, 2017.

**[H4] B. Baliś**, R. Brzoza-Woch, M. Bubak, M. Kasztelnik, B. Kwolek, P. Nawrocki, P. Nowakowski, T. Szydlo, K. Zielinski, “Holistic approach to management of IT infrastructure for environmental monitoring and decision support systems with urgent computing capabilities,” Future Generation Computer Systems, 2016, in press.

**[H5] B. Baliś**, B. Kowalewski, and M. Bubak, “Real-time Grid monitoring based on complex event processing,” Future Generation Computer Systems 27(8): 1103–1112, 2011.

**[H6]** M. Malawski, A. Gajek, A. Zima, **B. Baliś**, K. Figiela. „Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions.” *Future Generation Computer Systems*, 2017. In Press.

# Bibliografia (2)


[**H7] B. Baliś**, K. Figiela, M. Malawski, M. Pawlik, M. Bubak, “A lightweight approach for deployment of scientific workflows in cloud infrastructures,” in PPAM 2015, Lecture Notes in Computer Science, vol. 9573. Springer, 2016, pp. 281–290.

**[H8] B. Baliś**, K. Figiela, M. Malawski, and K. Jopek, “Leveraging workflows and clouds for a multi-frontal solver for finite-element meshes,” Procedia Computer Science, vol. 51, pp. 944–953, 2015. Proc. International Conference on Computational Science, ICCS 2015.

**[H9] B. Baliś**, M. Kasztelnik, M. Malawski, P. Nowakowski, B. Wilk, M. Pawlik, and M. Bubak, “Execution management and efficient resource provisioning for flood decision support,” Procedia Computer Science, vol. 51, pp. 2377–2386, 2015. Proc. International Conference on Computational Science, ICCS 2015.

**[H10] B. Baliś**, “Increasing Scientific Workflow Programming Productivity with HyperFlow,” in Proc. 9th Workshop on Workflows in Support of Large-Scale Science, WORKS ’14. IEEE Press, 2014, pp. 59–69.

**[H11] B. Baliś**, “Hypermedia workflow: a new approach to data-driven scientific workflows,” in WORKS ’12, High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion. IEEE, 2012, pp. 100–107.

**[H12] B. Baliś**, M. Kasztelnik, M. Bubak, T. Bartynski, T. Gubala, P. Nowakowski, and J. Broekhuijsen. The UrbanFlood Common Information Space for Early Warning Systems. *Procedia Computer Science*, 4:96-105, 2011. Proc. International Conference on Computational Science, ICCS 2011.

# DZIĘKUJĘ