

Porównanie implementacji archetypu dostępności czasowej

Artur Wojnar



Google
Developer
Groups

Jak zaimplementować system
rezerwacji czasowej zasobów,
zakładając duży ruch? 🤔



co to jest archetyp?



what_the_archetype_is|

Za Jimem Arlowem...

“ *A business archetype is a primordial thing that occurs consistently and universally in business domains and business software systems.*

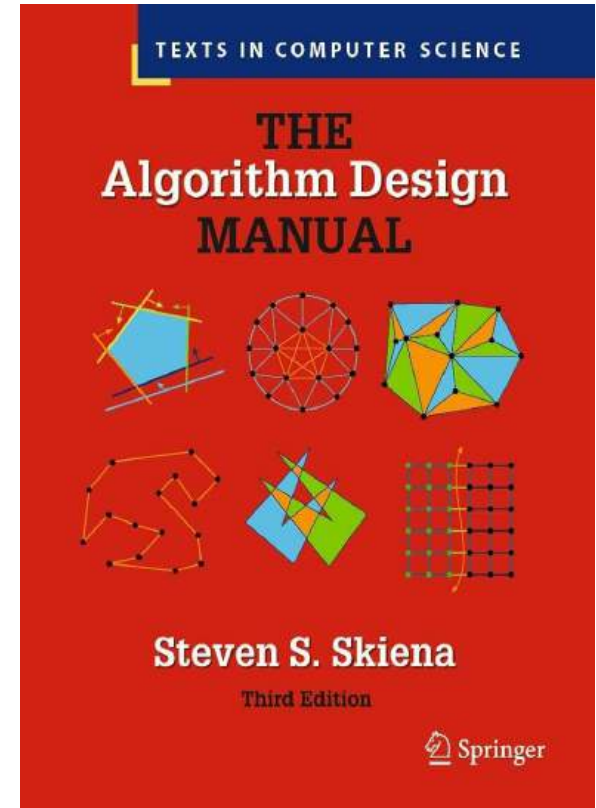
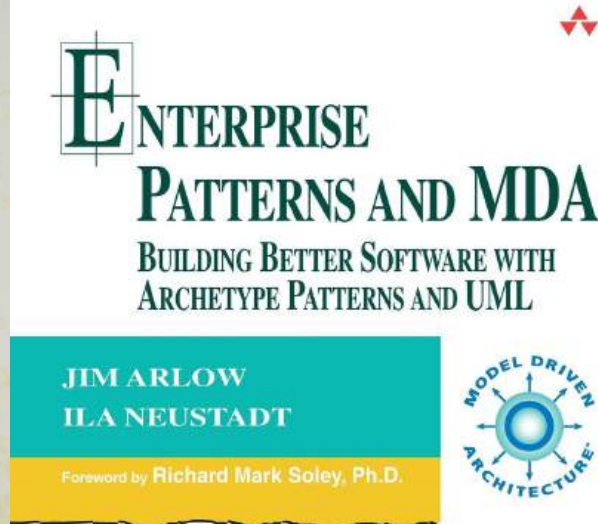
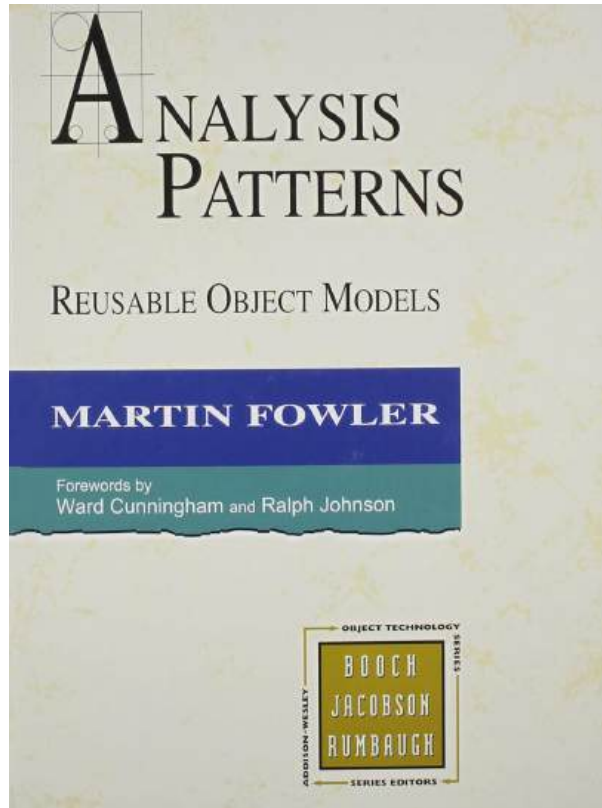
Archetypy często są wspólne dla wszystkich biznesów.

what_the_archetype_is|

Moje rozumienie czym jest archetyp:

Archetyp to zdefiniowane i opisane rozwiązanie dla powtarzającego się problemu biznesowego

archetype_sources |



archetypes_catalog|

**Party / Party
Relationship**









Pricing












Product



archetypes_catalog|

Party / Party Relationship 	Pricing 	Product 
Inventory 	Order(ing) 	Rules Engine 

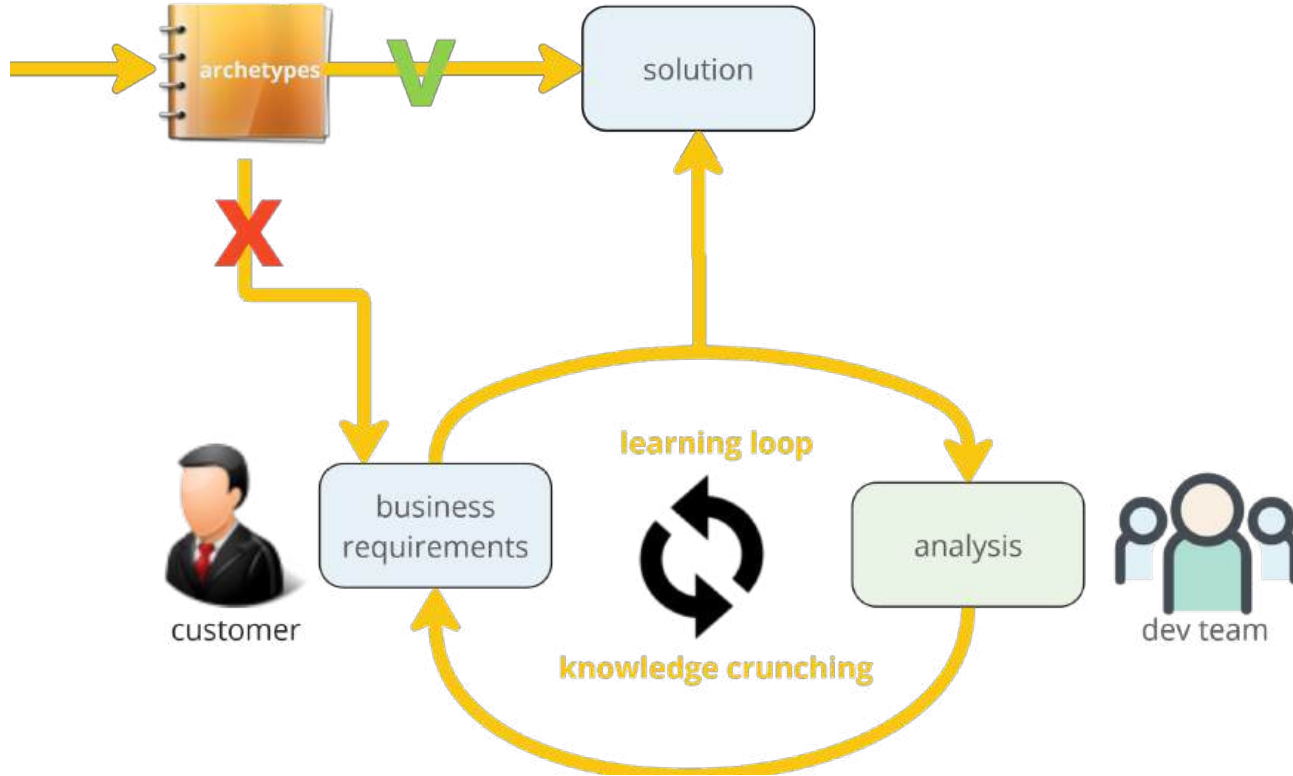
archetypes_catalog|

Party / Party Relationship 	Pricing 	Product 
Inventory 	Order(ing) 	Rules Engine 
Task 	Waitlist 	Knapsack problem 

why_archetypes_are_important|



why_archetypes_are_important|



why_archetypes_are_important|

- archetypy oszczędzają czas potrzebny na analizę

why_archetypes_are_important|

- archetypy oszczędzają czas potrzebny na analizę
- archetypy mają zdefiniowane ramy implementacji

why_archetypes_are_important|

- archetypy oszczędzają czas potrzebny na analizę
- archetypy mają zdefiniowane ramy implementacji
- **możesz udowodnić klientowi, że jesteś w stanie szybko dostarczyć wartość**

why_archetypes_are_important|

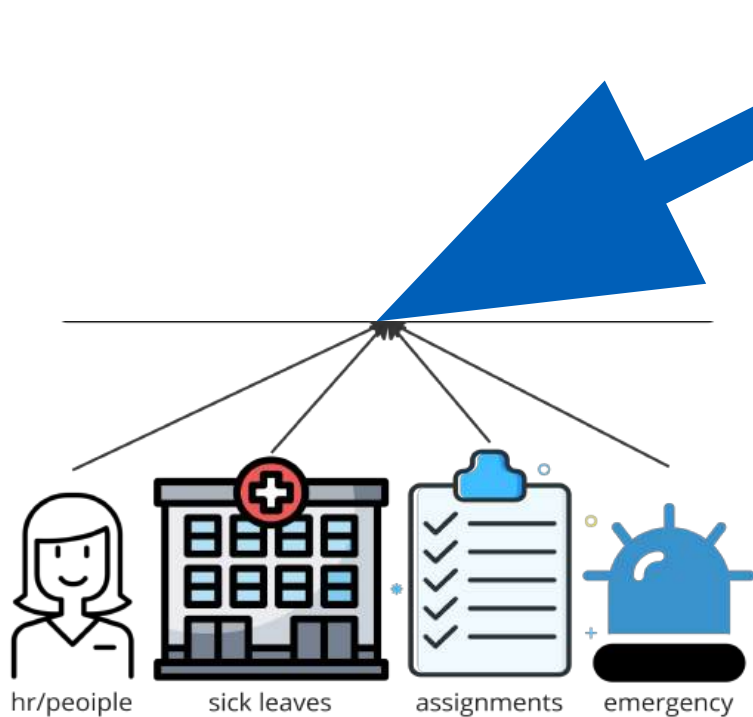
- archetypy oszczędzają czas potrzebny na analizę
- archetypy mają zdefiniowane ramy implementacji
- możesz udowodnić klientowi, że jesteś w stanie szybko dostarczyć wartość
- **jeden ze sposobów dzielenia się międzyprojektową wiedzą w organizacji**

archetyp dostępności

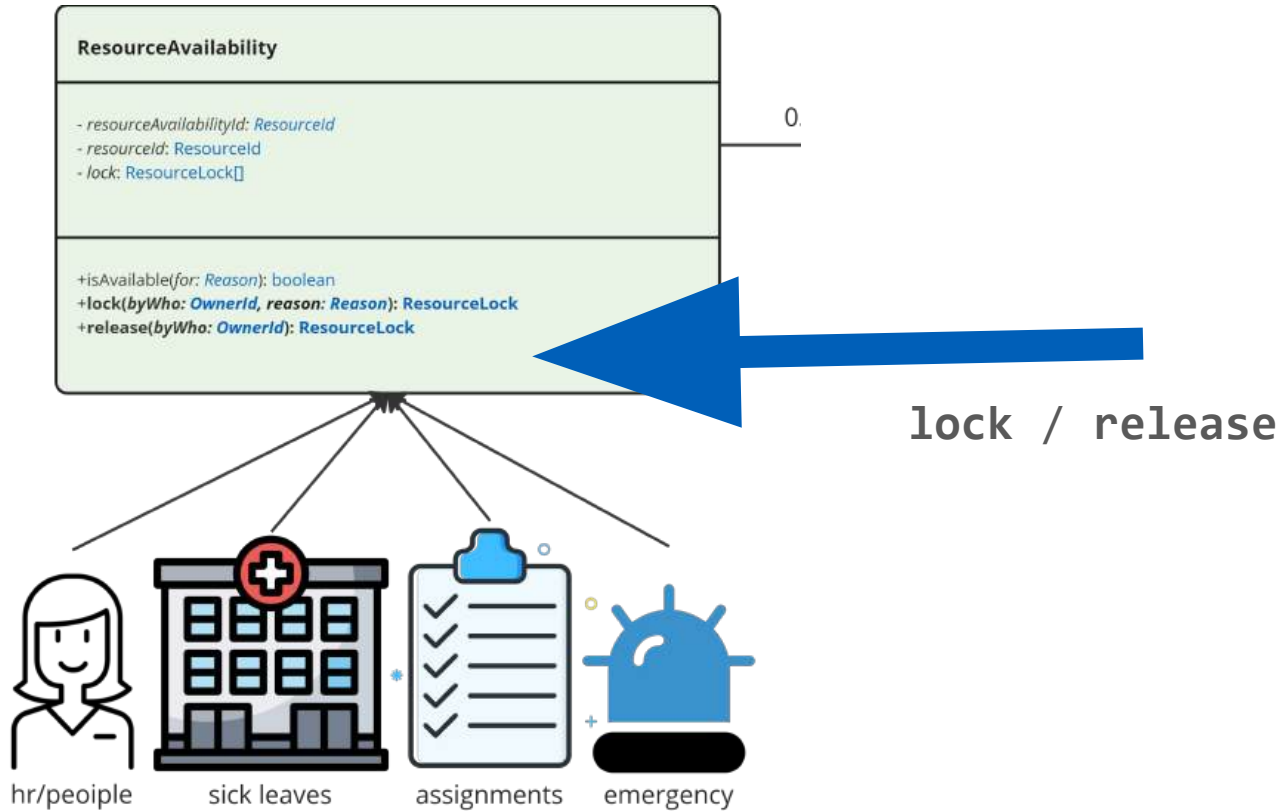


availability_archetype|

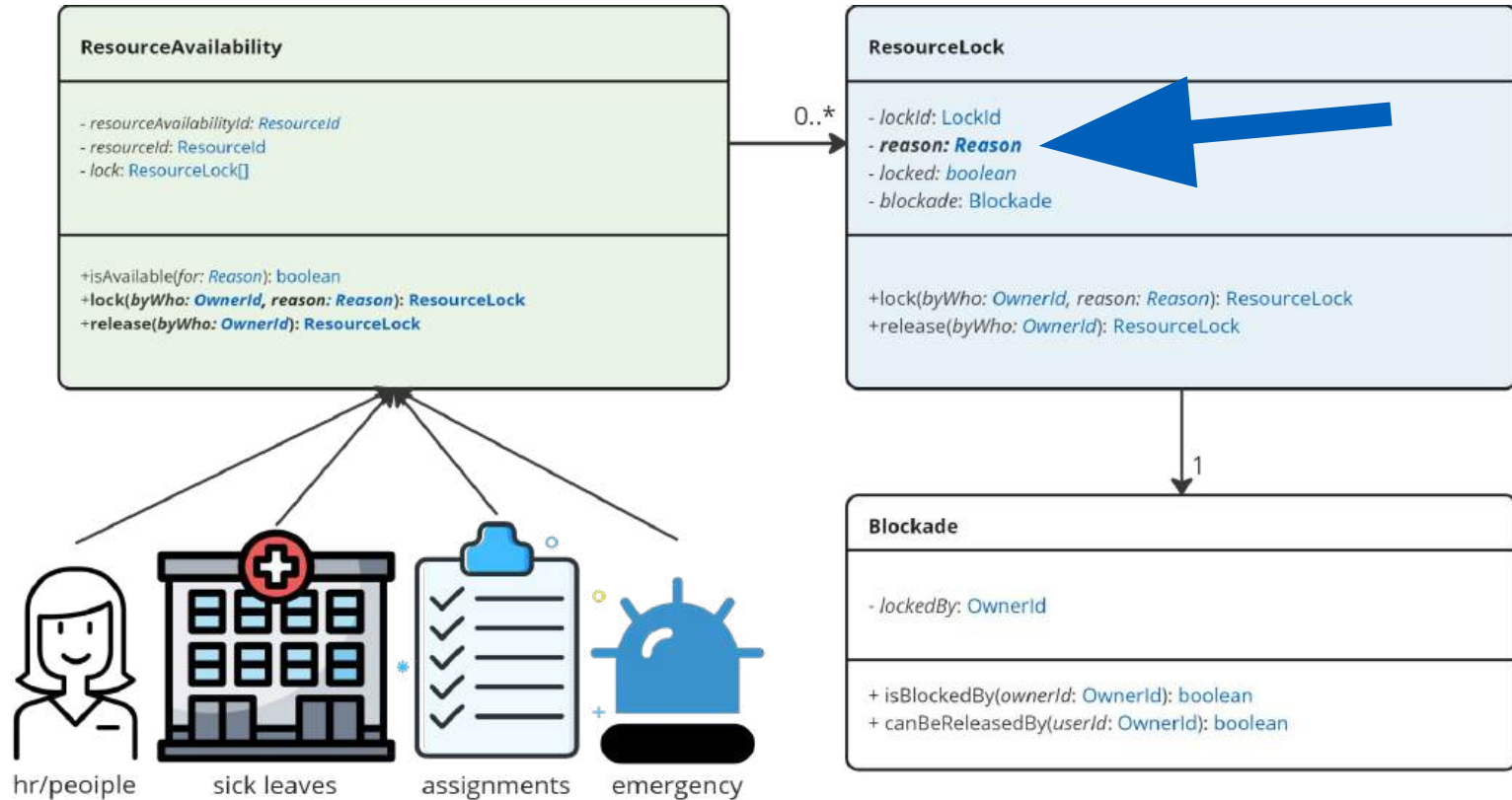
Jaki jest cel?



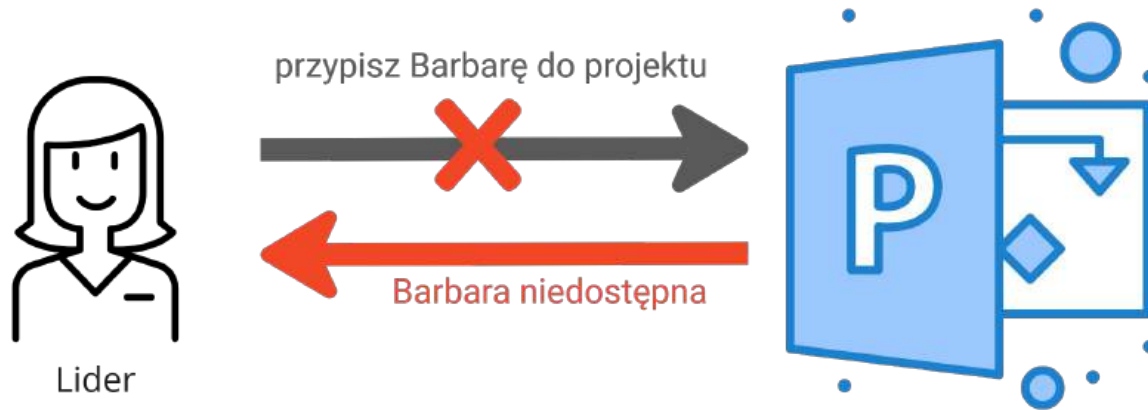
availability_archetype|



availability_archetype|



availability_archetype|

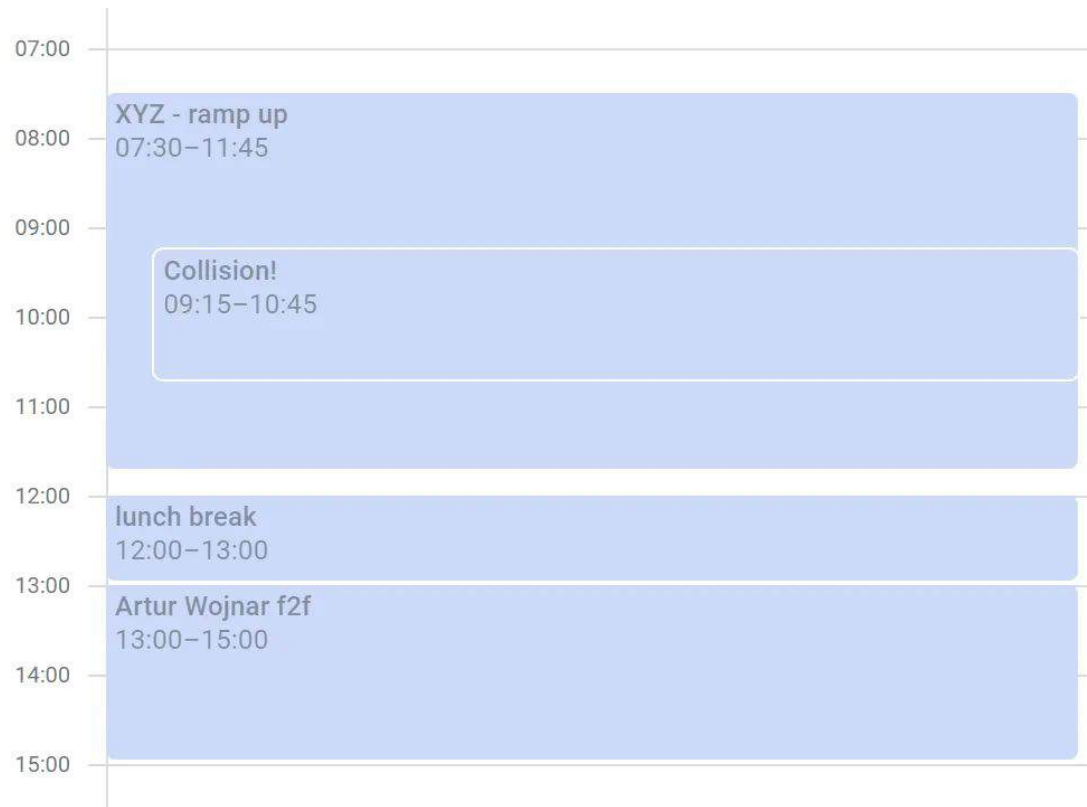


dostępność czasowa

UN	MON	TUE	WED	THU
		1	2	3
6	7	8	9	10
3	14	15	16	17

~ SIGHTS ~
ONE DOWN, 29 TO GO.

time_availability_archetype|



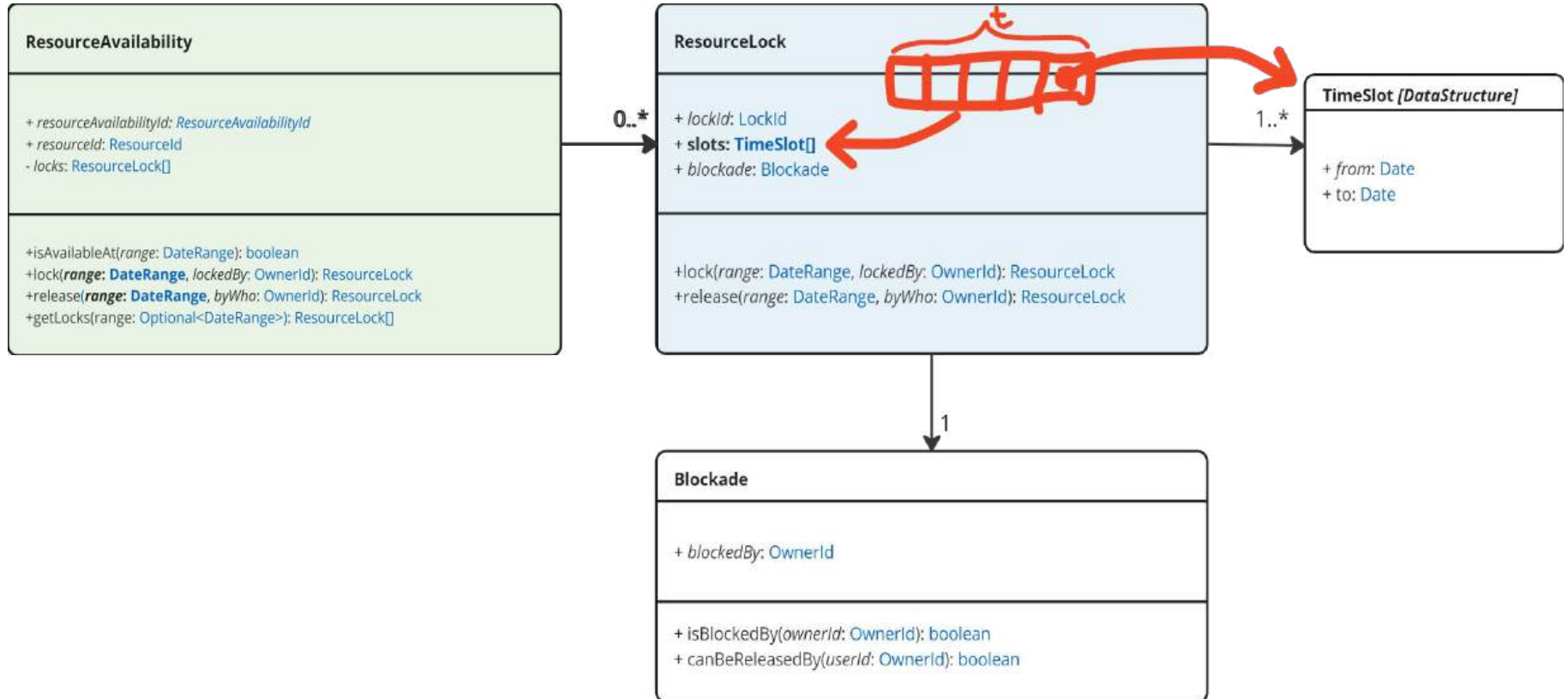
invariants|

1. nie można zarezerwować danego zasobu w dwóch nachodzących na siebie okresach
2. rezerwację czasową można zwolnić
3. zwolniony okres może być ponownie zajęty
4. minimalny okres rezerwacji to 15 minut (dla przykładu)

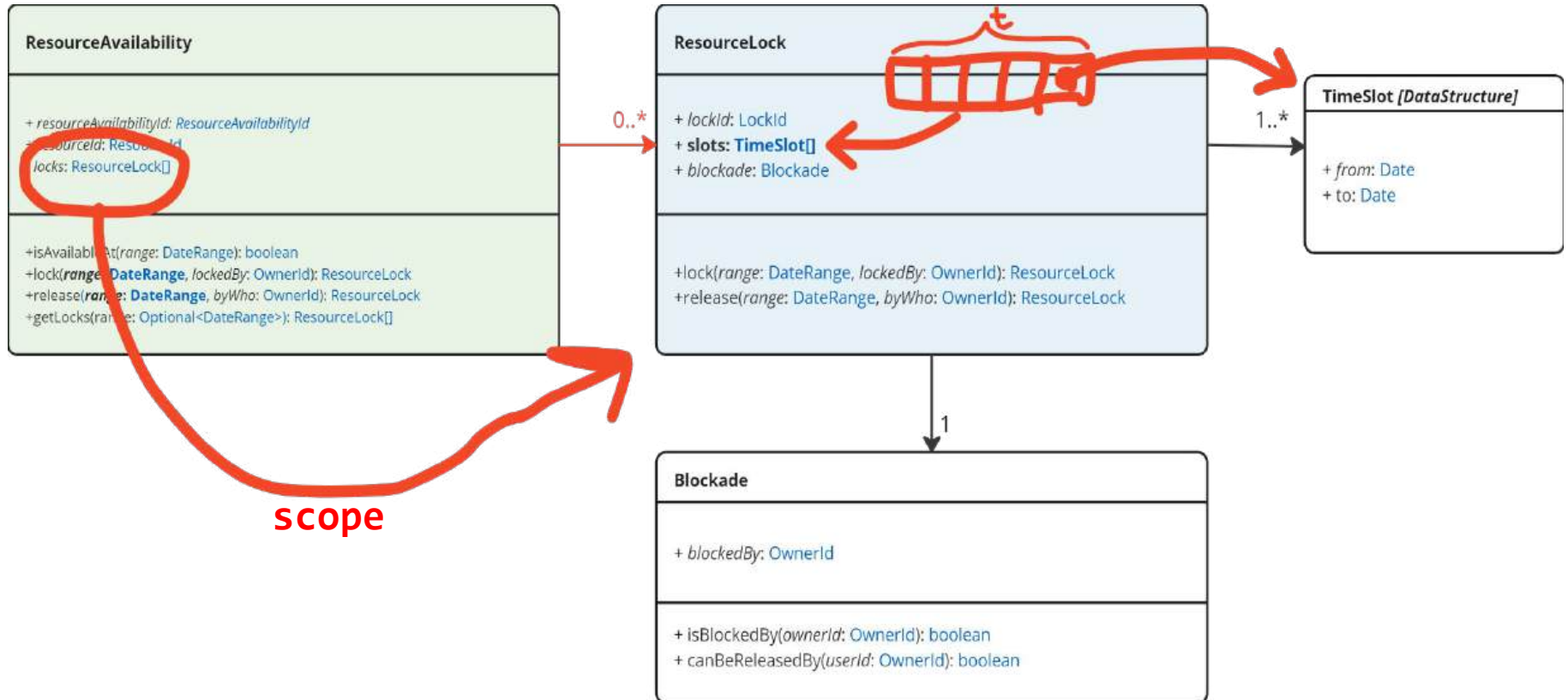
time_availability_archetype_uml |



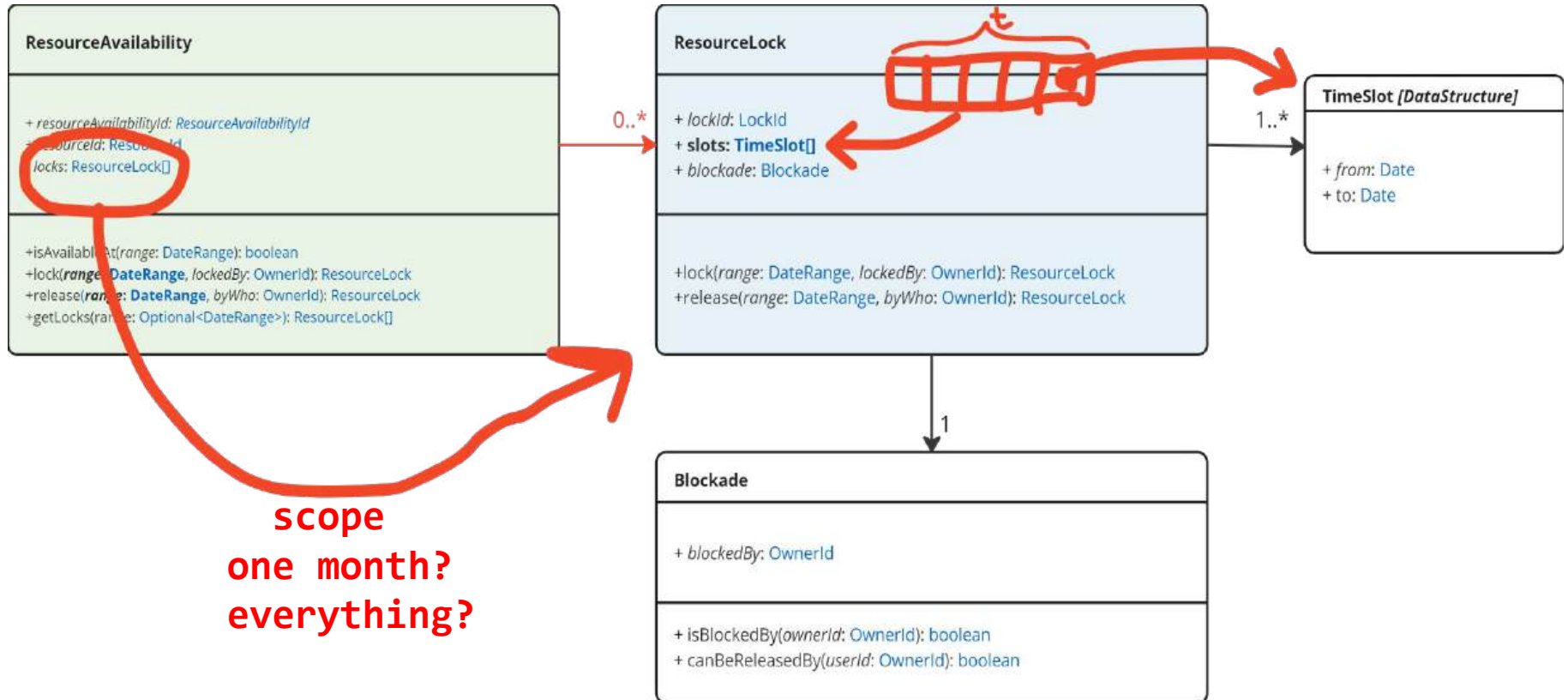
time_availability_archetype_uml |



time_availability_archetype_um1 |



time_availability_archetype_um1 |



how_to_find_the_availability|

Czy użytkownik
jest dostępny?

Czy użytkownik
ma czas?

Czy możemy
przydzielić X do
Y?

Czy *zasób* jest
dostępny?

Czy mamy zasób
na stanie?

Czy jest zasób
w magazynie?

naiwna implementacja



by_the_book_implementation|



```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```

by_the_book_implementation|


```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```

by_the_book_implementation|

```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```


by_the_book_implementation|

```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```



by_the_book_implementation|

```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```

by_the_book_implementation|

```
class ResourceAvailabilityService {  
  constructor(private _repo: ResourceAvailabilityRepo) {}  
  
  async lock(resourceId: ResourceId, requestorId: OwnerId, reason: Reason, lockFor: DateTimeRange) {  
    // getting data for the aggregate  
    const locks = await this._repo.find(resourceId)  
  
    // instantiating the aggregate  
    const resourceAvailability = new ResourceAvailability(resourceId, locks)  
  
    // changing the aggregate  
    resourceAvailability.lock(requestorId, reason, lockFor)  
  
    // persisting the aggregate  
    await this._repo.save(resourceAvailability)  
  }  
}
```

problem?

jak inaczej można podejść do tematu?



aggregate_revision|

“ *Domain-Driven Design has the Aggregate pattern to **ensure consistency** and to define **transactional concurrency boundaries** for object graphs*

Patterns, Principles and Practices of Domain-Driven Design
by Scott Millett and Nick Tune
ISBN 978-1-118-71470-6
page 320, chapter 14

“ *In any system with **persistent storage** of data, there must be a scope for a **transaction** that changes data, and a way of **maintaining the consistency** of the data (that is, maintaining its invariants)*

Domain-Driven Design: Tackling Complexity in the Heart of Software
by Eric Evans
ISBN 978-83-283-9184-0
page 76, chapter 6

our_implementation_paradigm|








1. OOP
2. FP
3. PP

our_implementation_paradigm|

1. OOP
2. FP
3. PP
4. ...and the winner is.... 🎉🎉🎉 SQL 🎉🎉🎉!



three_solutions|

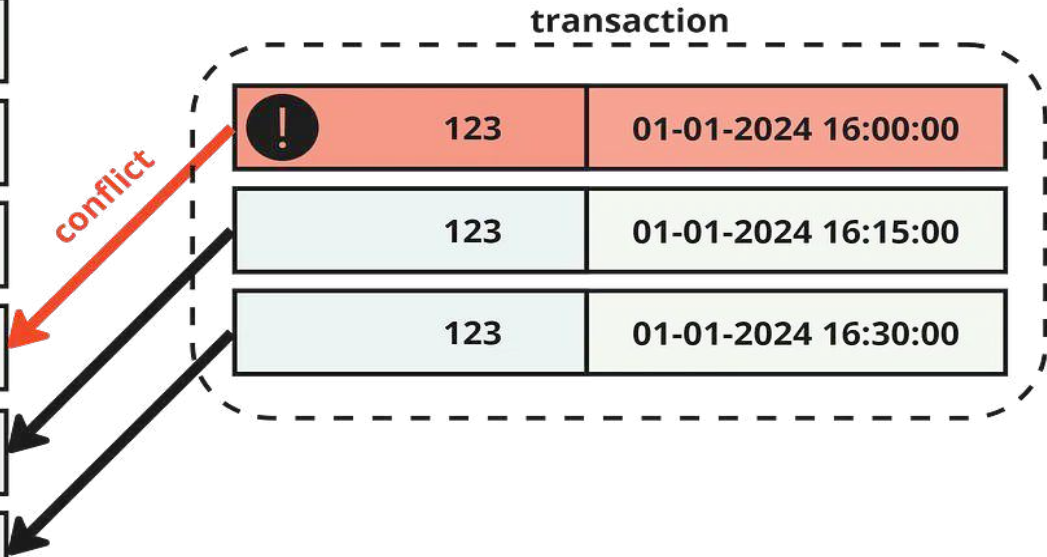
1. `timeslots` solution   
2. “`select`” solution  
3. “`gist`” solution  

timeslots solution



timeslot_solution|

resourceId	startTime
123	01-01-2024 15:00:00
123	01-01-2024 15:15:00
123	01-01-2024 15:30:00
123	01-01-2024 15:45:00
123	01-01-2024 16:00:00



timeslot_solution_scheme|

```
CREATE TABLE "TimeSlot" (  
    "id" SERIAL NOT NULL,  
    "requesterId" TEXT NOT NULL,  
    "resourceId" INTEGER NOT NULL,  
    "startTime" TIMESTAMP(3) NOT NULL,  
    "endTime" TIMESTAMP(3) NOT NULL,  
    "locked" BOOLEAN NOT NULL DEFAULT false,  
  
    CONSTRAINT "TimeSlot_pkey" PRIMARY KEY ("id")  
);  
  
CREATE UNIQUE INDEX "TimeSlot_resourceId_startTime" ON "TimeSlot"("resourceId", "startTime");  
...
```

timeslot_solution_lock_function|

```
CREATE OR REPLACE FUNCTION lock_timeslots(slots JSONB)
RETURNS VOID AS $$
DECLARE
    slot JSONB;
    v_locked BOOLEAN;
BEGIN
    FOR slot IN SELECT * FROM jsonb_array_elements(slots)
    LOOP
        INSERT INTO "TimeSlot" ("requesterId", "resourceId", "startTime", "endTime", "locked")
        VALUES (slot->>0, (slot->>1)::INTEGER, (slot->>2)::TIMESTAMPTZ, (slot->>3)::TIMESTAMPTZ, True)
        ON CONFLICT ("resourceId", "startTime")
        DO UPDATE SET "startTime" = EXCLUDED."startTime",
            "endTime" = EXCLUDED."endTime",
            "requesterId" = EXCLUDED."requesterId",
            "locked" = True
        WHERE "TimeSlot"."locked" = False
        RETURNING "TimeSlot"."locked" INTO v_locked;

        IF NOT FOUND THEN RAISE EXCEPTION 'CONFLICT'; END IF;
    END LOOP;
END $$ LANGUAGE plpgsql;
```

“select” solution



select_solution_lock_function|

```
CREATE OR REPLACE FUNCTION upsert_select_reservation(p_requester_id TEXT, p_resource_id INTEGER, p_start_time TIMESTAMPTZ, p_end_time TIMESTAMPTZ)
RETURNS VOID AS $$
DECLARE
    v_count INTEGER;
BEGIN
    -- Check for an existing timeslot that conflicts with the provided time range
    SELECT 1 INTO v_count
    FROM "TimeSlot2" t
    WHERE t."resourceId" = p_resource_id
        AND t."startTime" < p_end_time
        AND t."endTime" > p_start_time
        AND t."deleted" = FALSE
    LIMIT 1
    FOR UPDATE;

    -- If a conflicting timeslot is found, raise an exception
    IF FOUND THEN RAISE EXCEPTION 'CONFLICT on %-% (%)', p_start_time, p_end_time, p_resource_id; END IF;

    -- If no conflicts, insert the new timeslot
    INSERT INTO "TimeSlot2" ("requesterId", "resourceId", "startTime", "endTime", "deleted")
    VALUES (p_requester_id, p_resource_id, p_start_time, p_end_time, FALSE);
END;
$$ LANGUAGE plpgsql;
```

select_solution_scheme|

```
CREATE TABLE "TimeSlot2" (  
    "id" SERIAL NOT NULL,  
    "requesterId" TEXT NOT NULL,  
    "resourceId" INTEGER NOT NULL,  
    "startTime" TIMESTAMP(3) NOT NULL,  
    "endTime" TIMESTAMP(3) NOT NULL,  
    "deleted" BOOLEAN NOT NULL DEFAULT true,  
  
    CONSTRAINT "TimeSlot2_pkey" PRIMARY KEY ("id")  
);  
  
CREATE UNIQUE INDEX "TimeSlot2_index" ON "TimeSlot2"("resourceId", "startTime", "endTime",  
"deleted");  
...
```

“gist” solution



gist_solution|

```
CREATE EXTENSION IF NOT EXISTS btree_gist;
CREATE TABLE "timeslot3" (
    "id" SERIAL NOT NULL,
    "requesterid" TEXT,
    "resourceid" INTEGER,
    "date_range" tsrange,
    "deleted" BOOLEAN,
    CONSTRAINT "timeslot3_pkey" PRIMARY KEY ("id")
);

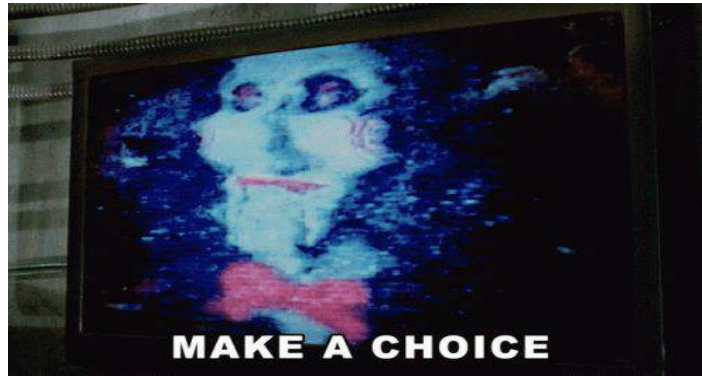
ALTER TABLE "timeslot3"
ADD CONSTRAINT "timeslot3_excl" EXCLUDE USING GIST
( "resourceid" WITH =, "date_range" WITH && ) WHERE ("deleted" IS FALSE);

CREATE UNIQUE INDEX "TimeSlot3_resourceId_daterange_deleted"
    ON "timeslot3"("requesterid", "resourceid", "date_range", "deleted");
. . .
```

gist_solution|

```
async lock({ requesterId, resourceId, date_range }: TimeSlot) {  
  await this._sql`  
    INSERT INTO "timeslot3" ("requesterid", "resourceid", "date_range", "deleted")  
    VALUES (  
      ${requesterId}::text,  
      ${resourceId}::int,  
      tsrange(${date_range[0].toISOString()}, ${date_range[1].toISOString()}, '[]'),  
      False  
    )`  
}
```

porównanie

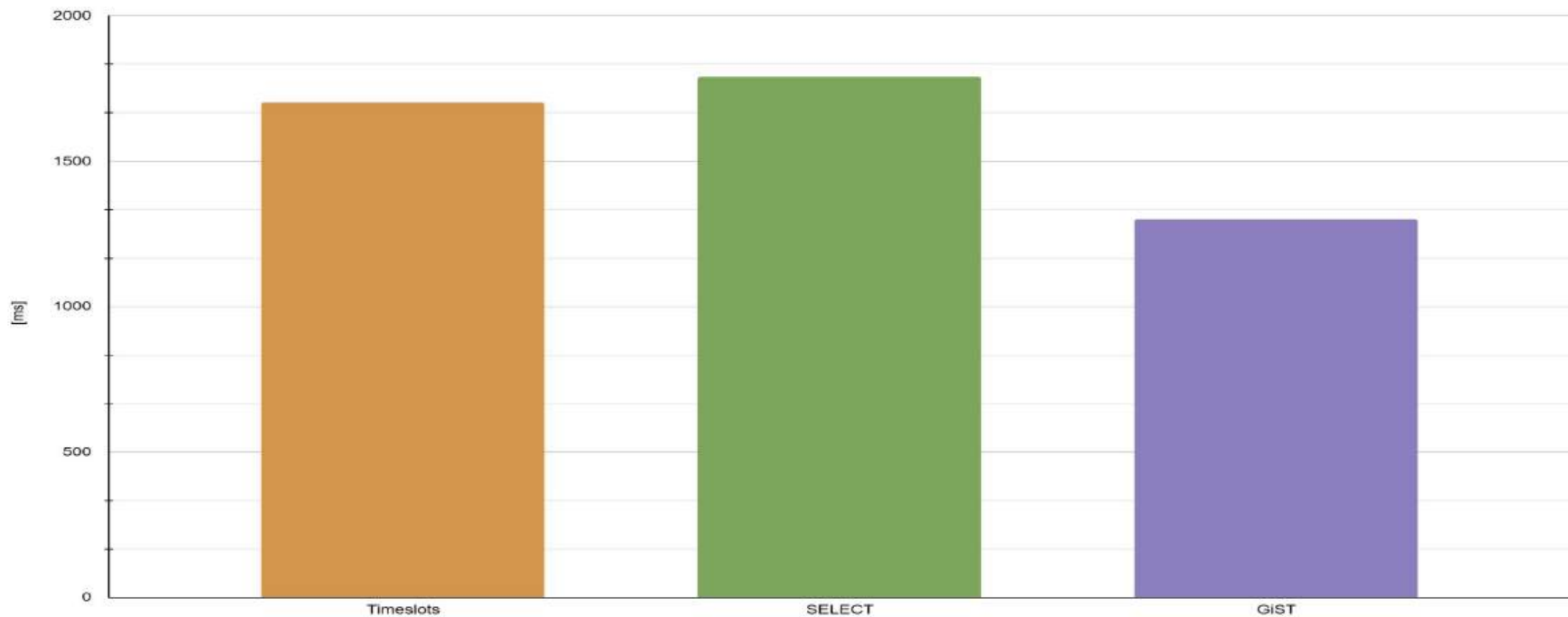


comparison|

Solution	Avg. on conflicts [ms]	Avg. on success [ms]	Avg. on unlocks [ms]	Avg. on relocks [ms]
Slots	2.3753	6.5949	4.7065	5.8578
Select	3.2314	3.4982	2.8718	3.1761
Gist	3.2314	3.4981	2.8718	3.1761

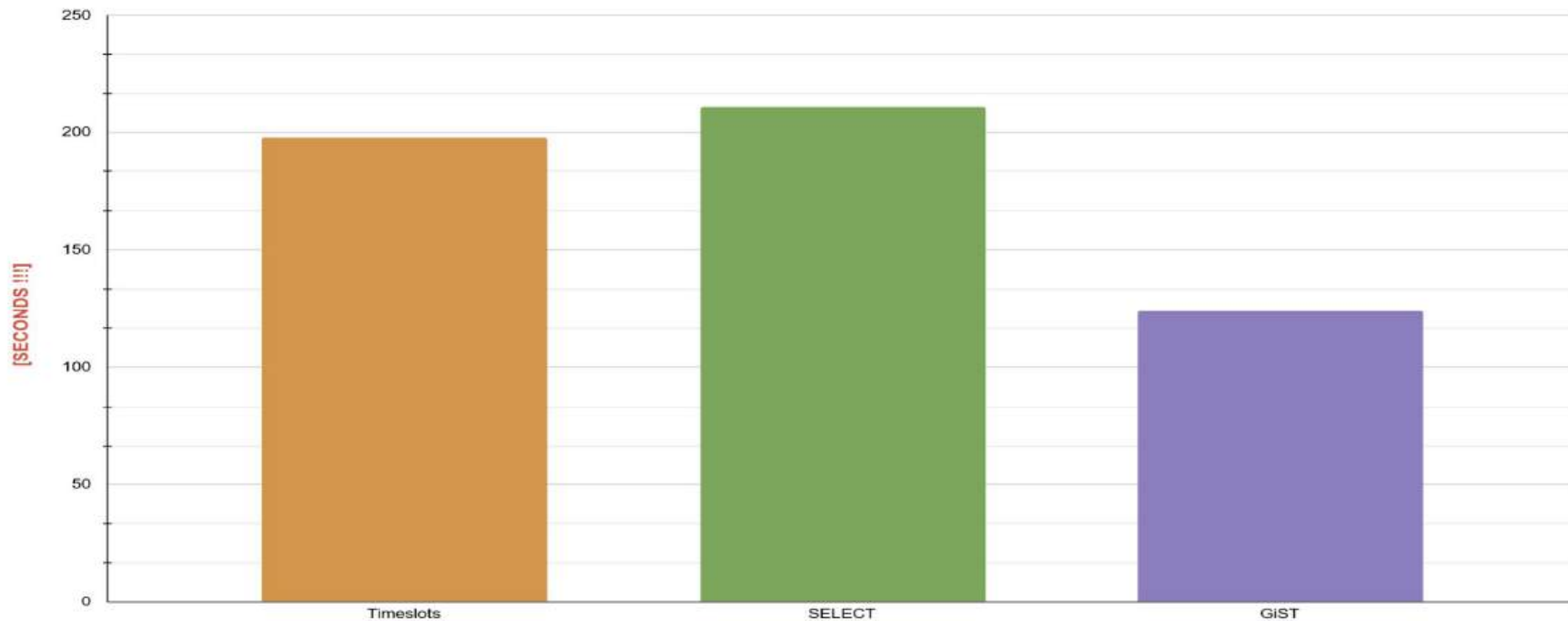
1000 zasobów. Każdy zasób ma 1000 blokad. Każda blokada trwa 30 godzin (120 time slotów)

comparison_concurrent_access|



10,000 nowych rezerwacji wykonanych jednocześnie

comparison_concurrent_access|

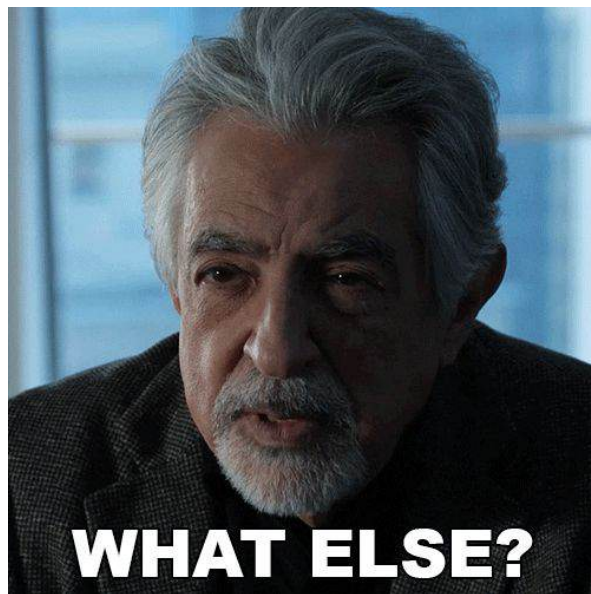


Milion nowych rezerwacji wykonanych jednocześnie

what_does_is_say|

- wszystkie rozwiązania są zbliżone pod kątem wydajności
- jakaś różnica pojawia się przy milionie jednoczesnych rezerwacji na korzyść GiST, ale w tym przypadku wyniki mogą być trudne do zaakceptowania

co jeszcze można zrobić?



mind_the_doamin|

- czego chce klient?
- jaka jest domena i jej specyfika?
- może jesteśmy w stanie uprościć nasz model?

simplified_time_availability|

Dr Barbara	08.10.24 / 08:00 / 15 min	Online consultation	1000
Dr Barbara	08.10.24 / 08:15 / 15 min	Online consultation	1001
Dr Barbara	08.10.24 / 08:30 / 15 min	Online consultation	1002
Dr Barbara	08.10.24 / 08:45 / 15 min	Online consultation	1003
Dr Barbara	08.10.24 / 09:30 / 15 min	Online consultation	1004

simplified_time_availability|

Dr Barbara	08.10.24 / 08:00 / 15 min	Online consultation	1000	booked by	version
Dr Barbara	08.10.24 / 08:15 / 15 min	Online consultation	1001		
Dr Barbara	08.10.24 / 08:30 / 15 min	Online consultation	1002		
Dr Barbara	08.10.24 / 08:45 / 15 min	Online consultation	1003		
Dr Barbara	08.10.24 / 09:30 / 15 min	Online consultation	1004		



Optimistic locking

time availability archetype -> availability archetype

what_does_is_say|

Czy klient potrzebuje spójności natychmiastowej?

Mechanizmy kontroli współbieżności (rodzaj blokady):

- Pesymistyczna
- Optymistyczna

what_does_is_say|

Czy klient potrzebuje spójności natychmiastowej?

Mechanizmy kontroli współbieżności (rodzaj blokady):

- Pesymistyczna
- Optymistyczna
- Kompensacja

what_does_is_say|

Czy klient potrzebuje spójności natychmiastowej?

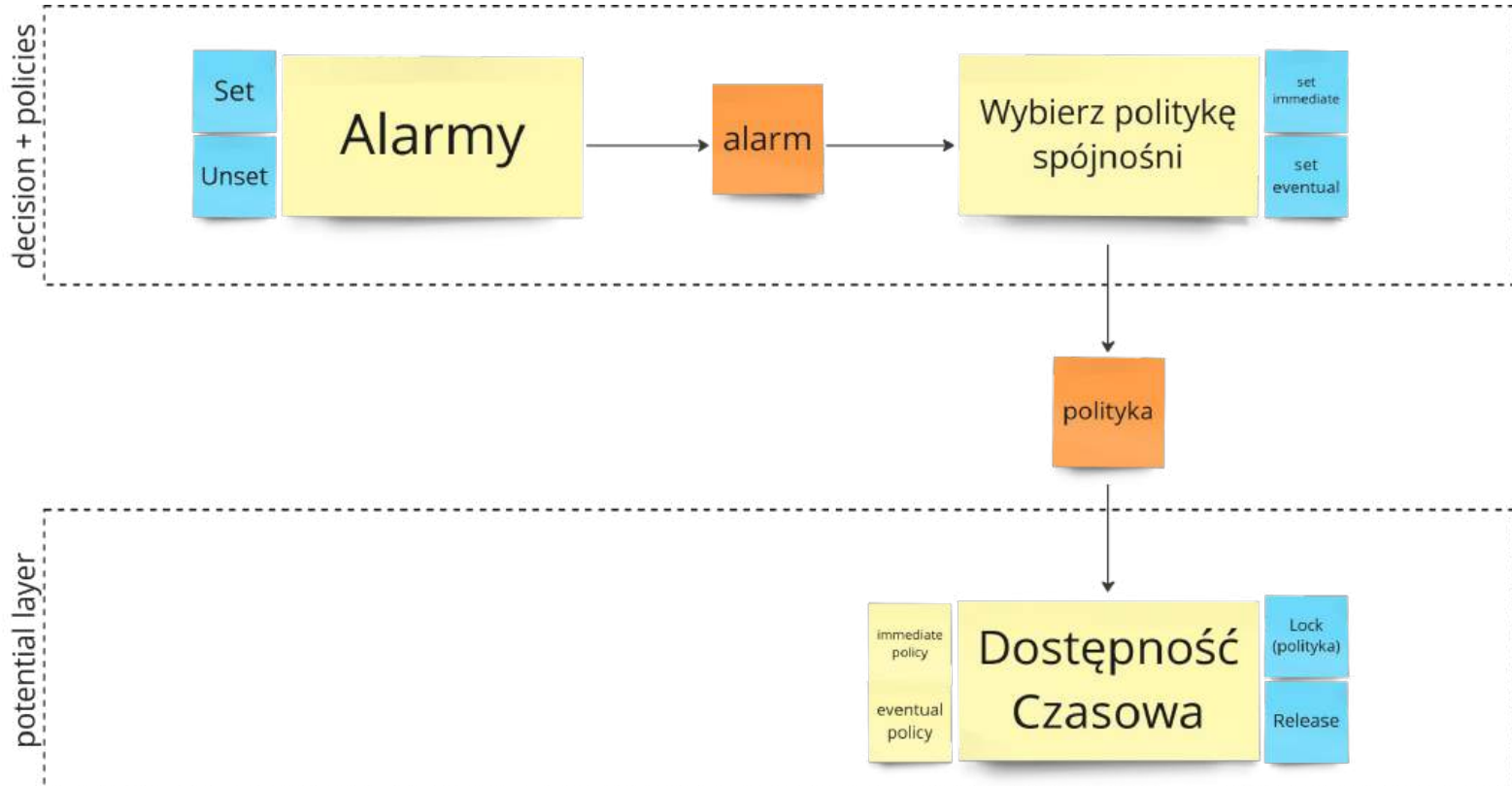
Mechanizmy kontroli współbieżności (rodzaj blokady):

- Pesymistyczna
- Optymistyczna
- **Kompensacja**
 - wykrywamy niespójność i ją naprawiamy
 - dopuszczamy eventual consistency
 - np. Write-Ahead Pattern

solution_based_on_compensating|



solution_based_on_compensating|



solutions_based_on_tooling|

- Elasticsearch
 - Łukasz Rynek, tech lider w *Znany Lekarz* (ang. *Dockplanner*, wiedzieliście?)
- Apache Flink
 - przetwarzanie utrwalonych strumieni danych

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- **obiektowa reprezentacja archetypu dostępności jest prosta**

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- obiektowa reprezentacja archetypu dostępności jest prosta
- ...ale jego implementacja może być wymagająca

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- obiektowa reprezentacja archetypu dostępności jest prosta
- ...ale jego implementacja może być wymagająca
- **najważniejszy jest kontekst**

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- obiektowa reprezentacja archetypu dostępności jest prosta
- ...ale jego implementacja może być wymagająca
- najważniejszy jest kontekst
- należy zwracać uwagę na ograniczenia jakie możemy zastosować

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- obiektowa reprezentacja archetypu dostępności jest prosta
- ...ale jego implementacja może być wymagająca
- najważniejszy jest kontekst
- najważniejszy jest biznes i ograniczenia jakie możemy zastosować
- **immediate consistency vs eventual consistency**

wrap-up |

- archetypy to generyczne rozwiązania dla wycinków biznesów
- wzorzec agregatu to tylko pojęcie i wytyczne - możemy je spełnić w różnych paradygmatach
- obiektowa reprezentacja archetypu dostępności jest prosta
- ...ale jego implementacja może być wymagająca
- najważniejszy jest kontekst
- najważniejszy jest biznes i ograniczenia jakie możemy zastosować
- immediate consistency vs eventual consistency
- narzędzia mogą pomóc w konkretnych przypadkach

about_me |

programista / solutions architect



<https://arturwojnar.dev>



<https://www.linkedin.com/in/artur-wojnar-a19349a6/>

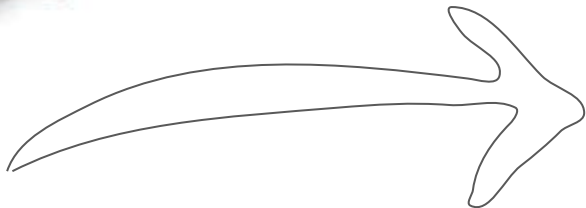


<https://revolve.healthcare/author/artur-wojnar>

references |

- [moje repo z przykładami](#)
- <https://arturwojnar.dev/comparing-time-availability-archetype>
- <https://bd90.pl/system-rezerwacji-w-5-minut/>
- <https://github.com/DomainDrivers>
- <https://www.postgresql.org/docs/14/gist.html>
- [compensating transaction](#)
- [write-ahead log pattern](#)
- [docplanner blog](#)
- [Patterns, Principles, and Practices of Domain-Driven Design](#)
- [Domain Modeling Made Functional by Scott Wlaschin](#)

pytania|



PROSZĘ BAAARDZOO
O UZUPEŁNIENIE ANKIETY



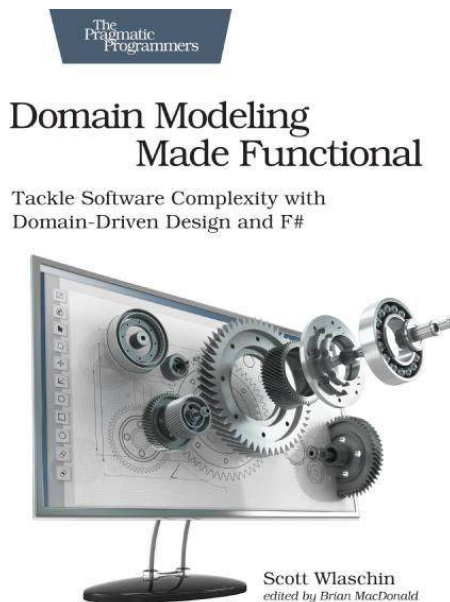
bonus: functional_aggregates|

- Decider Pattern
- Autor to Jeremie Chassaing
- Functional Event Sourcing Decider
- Maszyna stanów
- Komenda na wejściu, zdarzenia na wyjściu
- Zdarzenia zmieniają stan (a nie komenda)
- Budowniczy agregatów



bonus: functional_aggregates|

- Książka - Domain Modeling Made Functional
- Autor - Scott Wlaschin



jak archetypy łączą się z DDD?



archetypes_and_ddd|

business archetype → deep model

“ A “deep model” provides a lucid expression of the primary concerns of the domain experts and their most relevant knowledge while it sloughs off the superficial aspects of the domain.