KPHP community

KPHP FFI

Extending KPHP using foreign function interface API

Before we start...

~~Like & Subscribe~~

KPHP community chat: join today!

https://t.me/kphp_chat

Why I’m qualified to give this talk

- I added FFI support to KPHP compiler & runtime
- I created a rogue-like game with it

Topics for this talk

KPHP

FFI

SDL

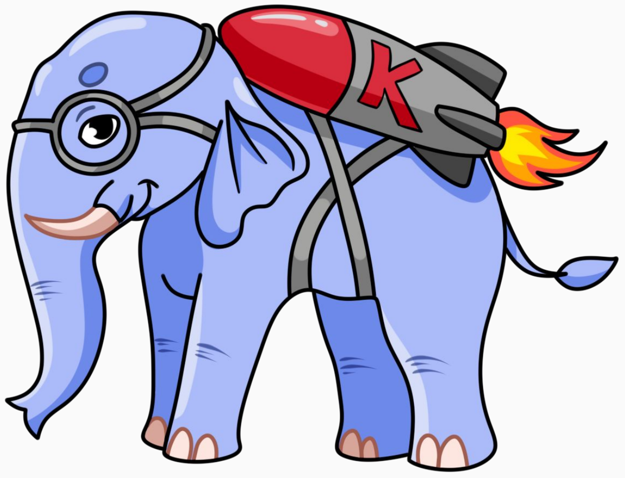
Rust bindings

Gamedev

What is KPHP?

What is KPHP?

A language with a cool mascot!



What is KPHP?

cough

What is KPHP?

- A PHP dialect that is type-safe

What is KPHP?

- A PHP dialect that is type-safe
- A compiler that creates executable binaries

What is KPHP?

- A PHP dialect that is type-safe
- A compiler that creates executable binaries
- An open source project

PHP FFI

PHP FFI

- A mechanism to call C functions from PHP

PHP FFI

- A mechanism to call C functions from PHP
- It’s similar to LuaJIT FFI and CPython FFI

PHP FFI

- A mechanism to call C functions from PHP
- It’s similar to LuaJIT FFI and CPython FFI
- Create PHP-extensions without C code!

So...

Why do we need FFI?

PHP FFI	KPHP FFI

The advantages of FFI

PHP FFI	KPHP FFI
● Pure PHP bindings for C	

The advantages of FFI

PHP FFI	KPHP FFI
• Pure PHP bindings for C • More portable than C ext	

The advantages of FFI

PHP FFI	KPHP FFI
● Pure PHP bindings for C ● More portable than C ext	● The only way to extend KPHP

The advantages of FFI

PHP FFI	KPHP FFI
● Pure PHP bindings for C ● More portable than C ext	● The only way to extend KPHP ● 100% compatible with PHP

The advantages of FFI

Write a C library wrapper once,

then use it from both PHP and KPHP!

Does KPHP support GD?

Does KPHP support GD?

Yes.

Does KPHP support GD?

Yes.

Use FFI.

```php
$gd = FFI::cdef('
  typedef struct gdImage gdImage;
  gdImage *gdImageCreate(int sx, int sy);
  void gdImageDestroy(gdImage *image);
', 'libgd.so');
```

```php
$img = $gd->gdImageCreate(32, 32);
$gd->gdImageDestroy($img);
```

Simple usage example

```
$gd = FFI::cdef('
  typedef struct gdImage gdImage;
  gdImage *gdImageCreate(int sx, int sy);
  void gdImageDestroy(gdImage *image);
', 'libgd.so');
```

FFI::cdef creates a FFI handle from a C string and loads associated shared (dynamic) library

Simple usage example

```
$gd = FFI::cdef('
    typedef struct gdImage gdImage;
    gdImage *gdImageCreate(int sx, int sy);
    void gdImageDestroy(gdImage *image);
', 'libgd.so');
```

C declarations string (like in a C header file)

Simple usage example

```
$gd = FFI::cdef('
  typedef struct gdImage gdImage;
  gdImage *gdImageCreate(int sx, int sy);
  void gdImageDestroy(gdImage *image);
', 'libgd.so');
```

ldconfig-compatible name for the library lookup

Simple usage example

$gd is our FFI library handle; it’s used to access C functions, types, variables and constants (enums, etc).

```
$img = $gd->gdImageCreate(32, 32);
$gd->gdImageDestroy($img);
```

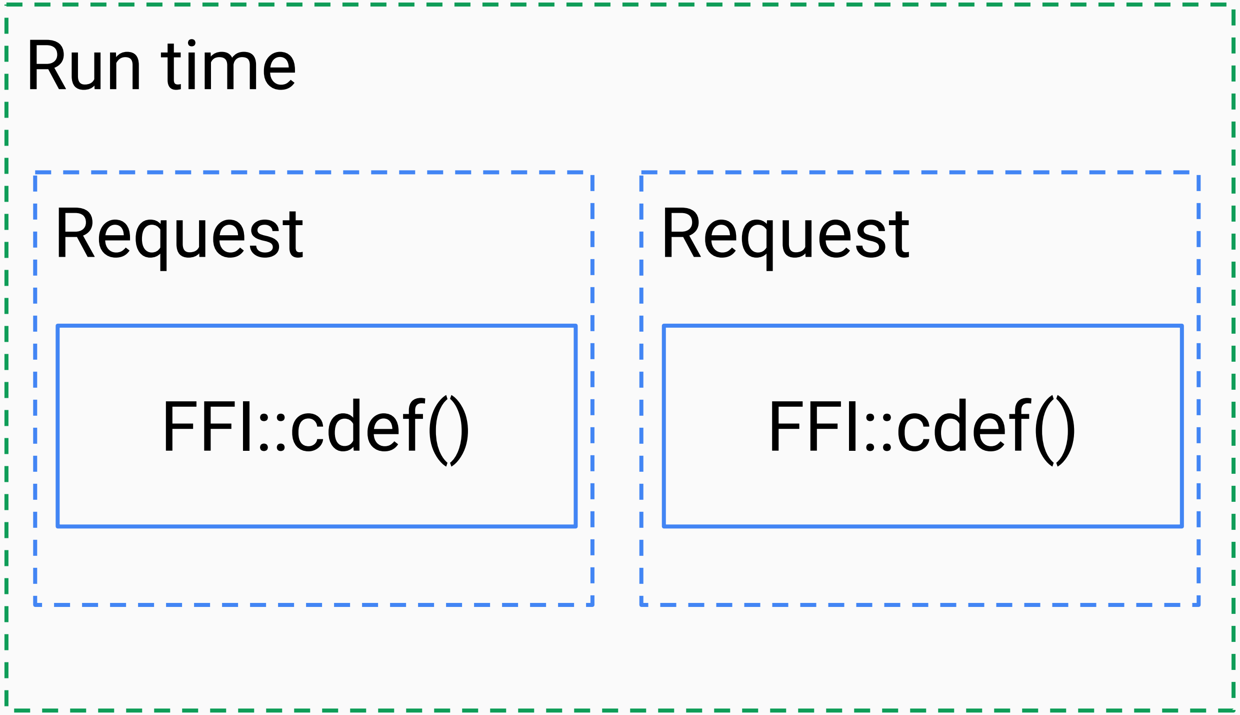
Simple usage example

gd.h

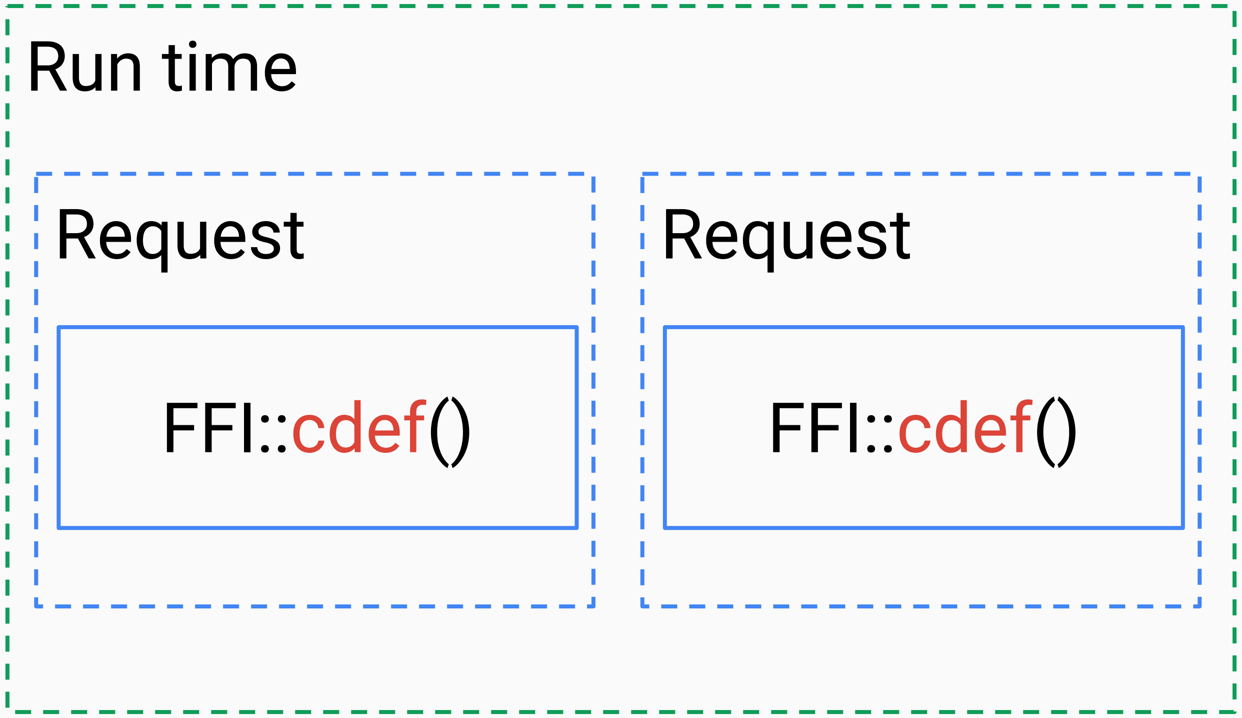
```c
#define FFI_LIB "libgd.so"

typedef struct gdImage gdImage;
gdImage *gdImageCreate(int sx, int sy);
void gdImageDestroy(gdImage *image);
```

```php
$gd = FFI::load(__DIR__ . '/gd.h');
```

FFI::load - load from a separate declarations file

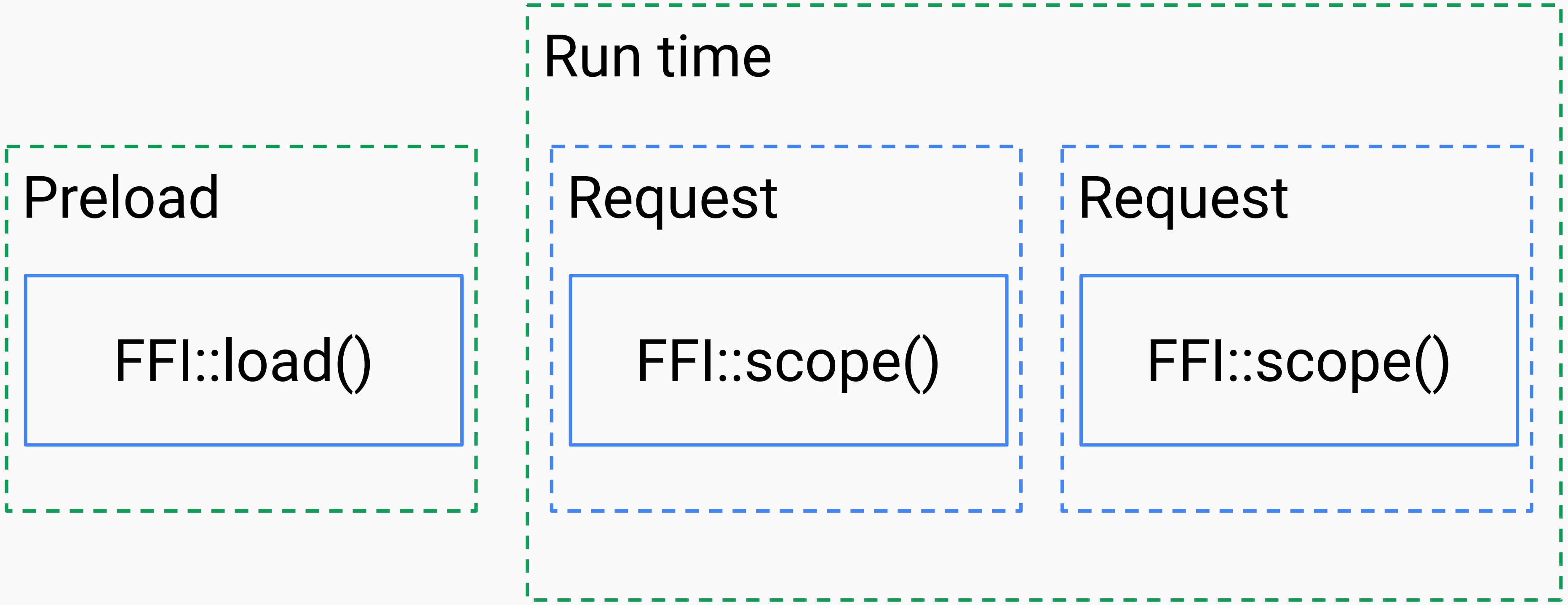


FFI::cdef usage scheme

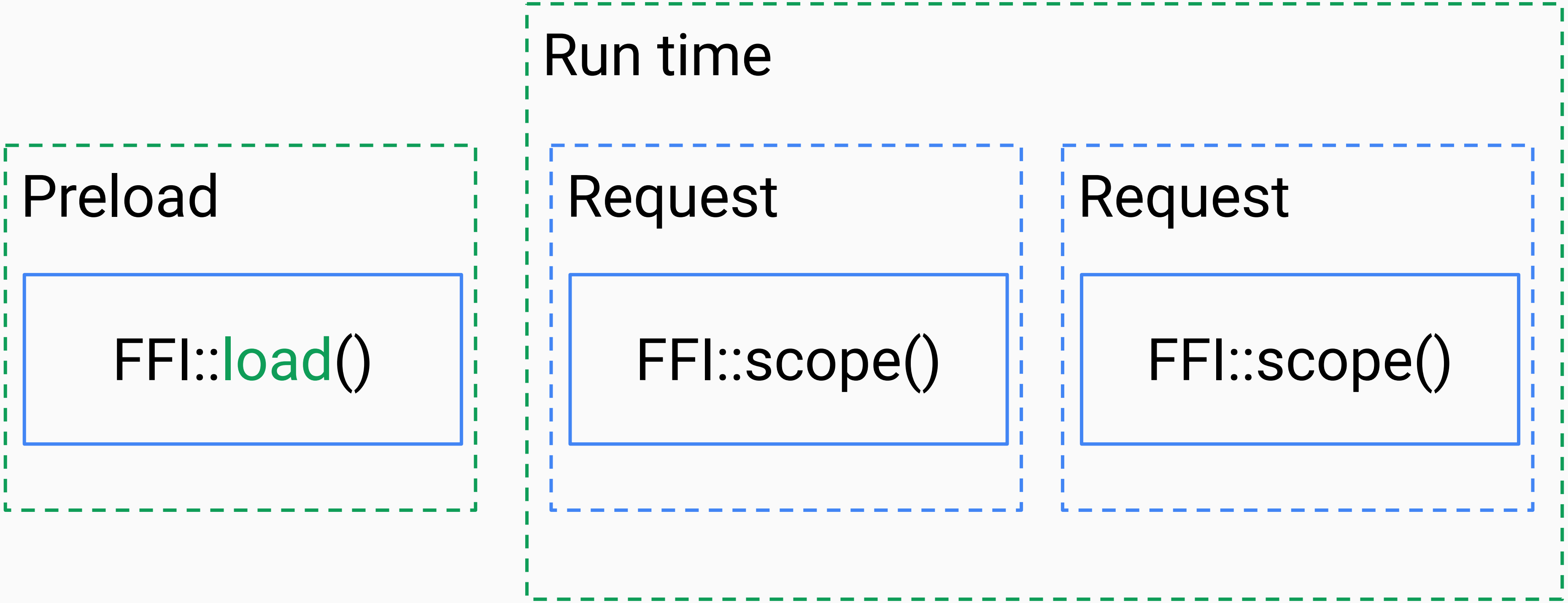


Bad! Parsing C declarations for every request

FFI::cdef usage scheme



FFI::load() + preload usage scheme



Good! Parsing C declarations only once

FFI::load() + preload usage scheme

Comparing PHP and KPHP load/cdef

PHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj
- parses C decls

KPHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj

PHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj
- parses C decls

KPHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj

KPHP doesn’t need FFI::scope() for performance

Comparing PHP and KPHP load/cdef

PHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj
- parses C decls

KPHP load/cdef

- loads shared libs
- fetches symbols
- creates a FFI obj

But we're using FFI::scope() for the type checking!

Comparing PHP and KPHP load/cdef

```c
#define FFI_SCOPE foo

struct Example { const char *s; int16_t i; };

char ffi_func(int16_t x, const char *s);
```

```php
$foo = FFI::scope("foo");
```

FFI scope object

```c
#define FFI_SCOPE foo

struct Example { const char *s; int16_t i; };

char ffi_func(int16_t x, const char *s);
```

```php
// OK
$ex = $scope->new("struct Example");
```

FFI scope object

```
#define FFI_SCOPE foo

struct Example { const char *s; int16_t i; };

char ffi_func(int16_t x, const char *s);
```

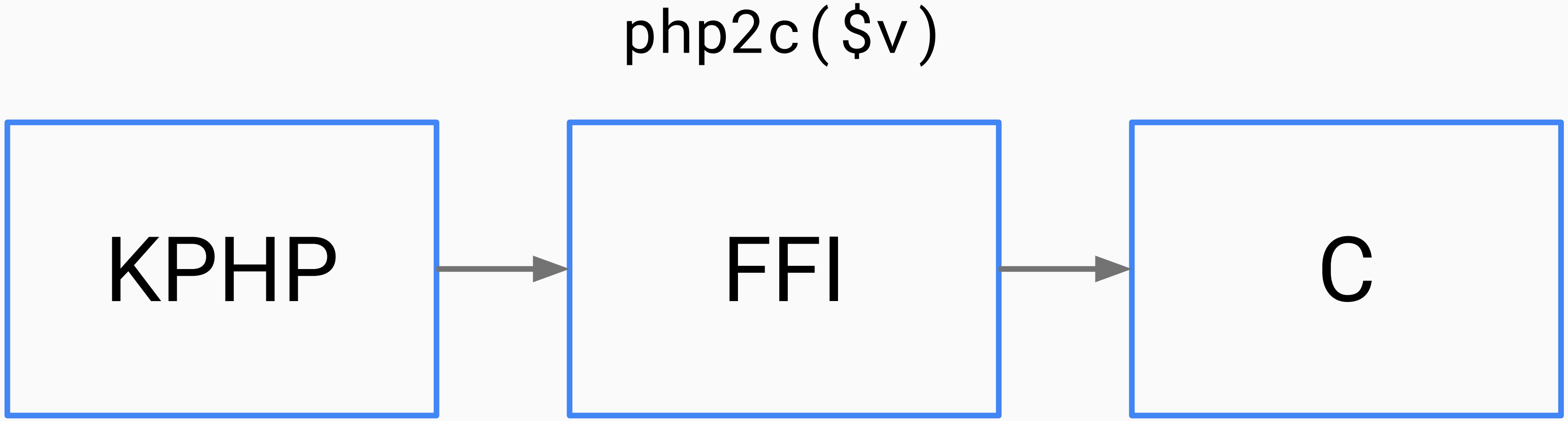
```


// OK
$scope->ffi_func(10, 'hello');


```

FFI scope object

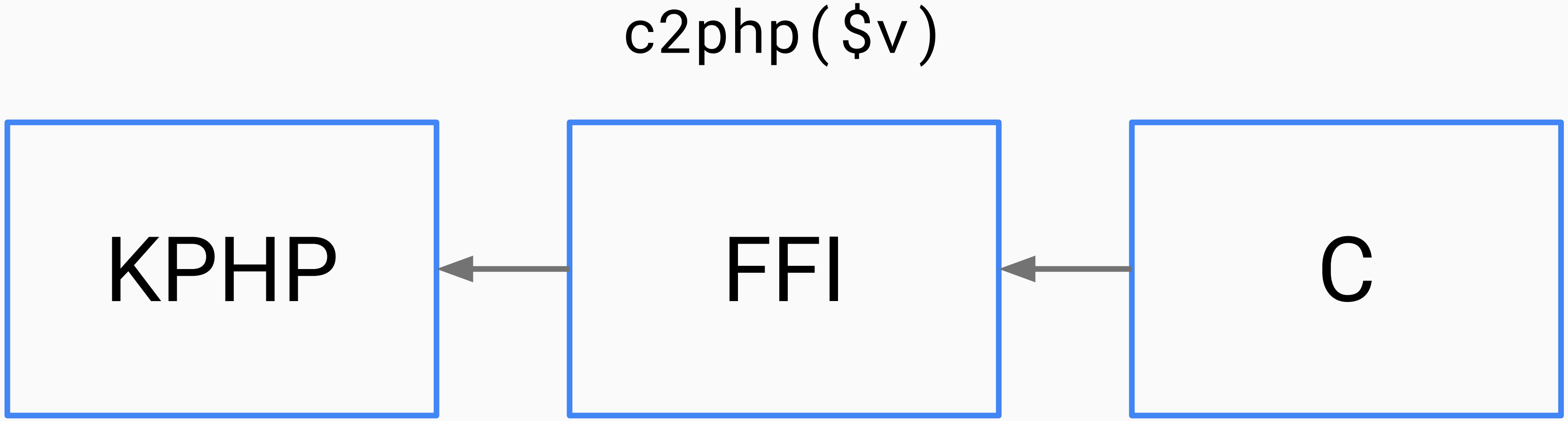
```
#define FFI_SCOPE foo

struct Example { const char *s; int16_t i; };

char ffi_func(int16_t x, const char *s);
```

```
// ERROR (compile-time)
$scope->undefined_func();
```

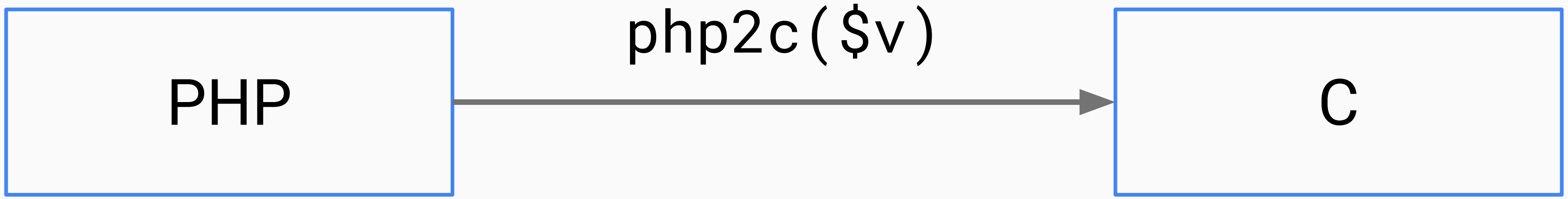
FFI scope object



Passing KPHP values as C func args



Mapping C func result to KPHP value



- Passing C function argument
- Assigning to C struct/union field
- Assigning to a pseudo cdata field

Auto conversions

PHP type	C type
int (long)	int8_t, int16_t, ...
float	float, double
bool	bool
string(1)	char
string	const char*

php2c conversions

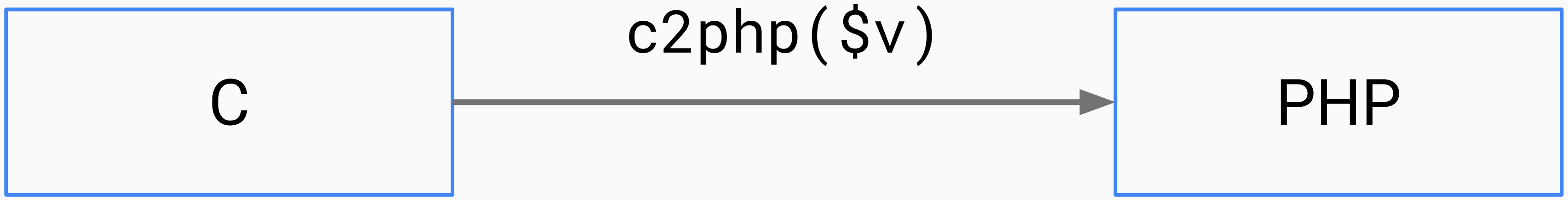
PHP type	C type
int (long)	int8_t, int16_t, ...
float	float, double
bool	bool
string(1)	char
string	const char*

Only for function arguments, but not struct field write

php2c conversions

PHP type	C type
CData<T>	T
FFI::addr(CData<T>)	T*

php2c conversions



- Assigning a (non-void) C function result
- Reading C struct/union field
- Reading C scalar "cdata" property
- Reading Scope property (enum values, etc)

Different conversion rules for call results and fields!

Auto conversions

C type	PHP type
int8_t, int16_t, ...	int
float, double	float
bool	bool
char	string(1)
const char*	string

c2php conversions

C type	PHP type
int8_t, int16_t, ...	int
float, double	float
bool	bool
char	string(1)
const char*	string

Only for function results, but not for struct field read

c2php conversions

C type	PHP type
T	CData<T>
T*	CData<T*>

php2c conversions

What is CData?

What is CData?

Types that can’t be represented as normal PHP types are wrapped into CData classes.

```php
/** @return ffi_cdata<example, struct Foo> */
function f() {
  $cdef = FFI::cdef('
    #define FFI_SCOPE "example"
    struct Foo { int x; };
  ');
  return $cdef->new('struct Foo');
}
```

CData

```
/** @return ffi_cdata<example, struct Foo> */
function f() {
  $cdef = FFI::cdef('
    #define FFI_SCOPE "example"
    struct Foo { int x; };
  ');
  return $cdef->new('struct Foo');
}
```

new(T) returns CData<T> typed object

CData

```
/** @return ffi_cdata<example, struct Foo> */
function f() {
  $cdef = FFI::cdef('
    #define FFI_SCOPE "example"
    struct Foo { int x; };
  ');
  return $cdef->new('struct Foo');
}
```

T is a type from associated FFI scope/cdef

CData

```php
/** @return ffi_cdata<example, struct Foo> */
function f() {
  $cdef = FFI::cdef('
    #define FFI_SCOPE "example"
    struct Foo { int x; };
  ');
  return $cdef->new('struct Foo');
}
```

PHP type hint expects both scope and type

FFI\CData

```
template<class T>
struct FFI_CData {
  T value;
}
```

```
ffi_cdata<scope,T>
```

FFI CData runtime representation

Can I do gamedev in KPHP?

Can I do gamedev in KPHP?

With things like SDL, you can!

SDL libraries

- libsdl2
- libsdl2_image
- libsdl2_mixer
- libsdl2_ttf

SDL libraries

- libsdl2 $\longrightarrow$ sdl.h
- libsdl2_image $\longrightarrow$ sdl_image.h
- libsdl2_mixer $\longrightarrow$ sdl_mixer.h
- libsdl2_ttf $\longrightarrow$ sdl_ttf.h

KPHP game with SDL

Part 1: creating GUI window

sdl.h

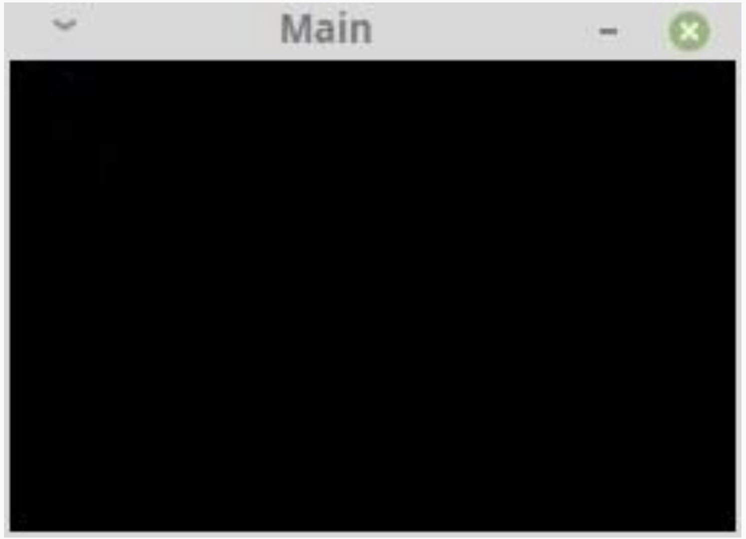
```c
#define FFI_SCOPE "sdl"
#define FFI_LIB "libSDL2-2.0.so"

typedef uint32_t Uint32;
typedef struct SDL_Window SDL_Window;

SDL_Window *SDL_CreateWindow(
  const char *title,
  int x, int y, int w, int h,
  Uint32 flags);
```

Creating our first header file for libsdl2

test.php

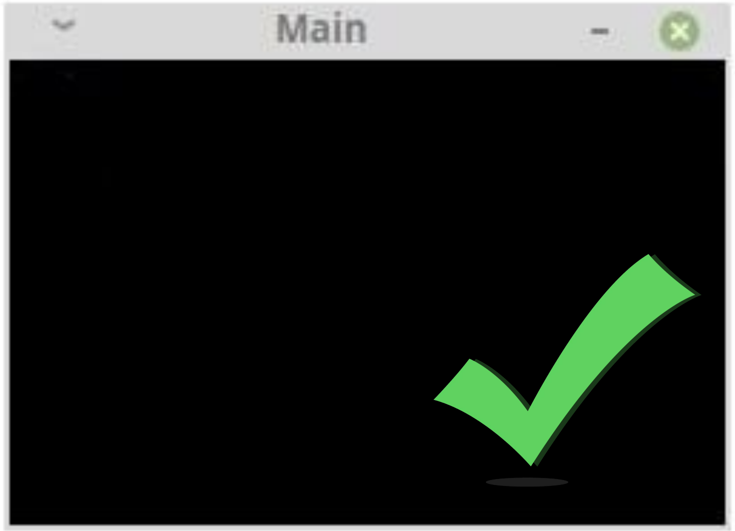
```php
\FFI::load('sdl.h');
$sdl = \FFI::scope('sdl');

$w = 640;
$h = 480;
$window = $sdl->SDL_CreateWindow(
   'Main', 0, 0, $w, $h, 0);
```

Creating our first header file for libsdl2


Main

How to create a video game using SDL

1.



2.

KPHP game with SDL

Part 2: creating event loop

Game.php

```php
while (true) {
   $this->processInputs($sdl);
   if ($this->exit) {
      break;
   }
   $this->processFrame($sdl);
   $sdl->delay(1000 / 60); // ~60 fps
}
```

Game event loop

Game.php

```php
while (true) {
    $this->processInputs($sdl);
    if ($this->exit) {
        break;
    }
    $this->processFrame($sdl);
    $sdl->delay(1000 / 60); // ~60 fps
}
```

Reading all incoming input events (key press, signals, etc).

Game event loop

Game.php

```php
while (true) {
   $this->processInputs($sdl);
   if ($this->exit) {
      break;
   }
   $this->processFrame($sdl);
   $sdl->delay(1000 / 60); // ~60 fps
}
```

If player pressed “esc” or quit signal is received, exit the event loop.

Game event loop

Game.php

```
while (true) {
   $this->processInputs($sdl);
   if ($this->exit) {
      break;
   }
   $this->processFrame($sdl);
   $sdl->delay(1000 / 60); // ~60 fps
}
```

Execute game logic: handle game frame for all objects (player, enemies, etc).

Game event loop

Game.php

```
while (true) {
   $this->processInputs($sdl);
   if ($this->exit) {
      break;
   }
   $this->processFrame($sdl);
   $sdl->delay(1000 / 60); // ~60 fps
}
```

Wait for the next frame.

Game event loop

Game.php

```php
$event = $sdl->newEvent();
while ($sdl->pollEvent($event)) {
   if ($event->type === EventType::QUIT) {
      $this->exit = true;
   } elseif ($event->type === EventType::KEYUP) {
      // handle key up event
   }
   // and so on...
}
```

Event handling

Game.php

```php
$event = $sdl->newEvent();
while ($sdl->pollEvent($event)) {
   if ($event->type === EventType::QUIT) {
      $this->exit = true;
   } elseif ($event->type === EventType::KEYUP) {
      // handle key up event
   }
   // and so on...
}
```

Creating an event object to fill.

Event handling

Game.php

```php
$event = $sdl->newEvent();
while ($sdl->pollEvent($event)) {
    if ($event->type === EventType::QUIT) {
        $this->exit = true;
    } elseif ($event->type === EventType::KEYUP) {
        // handle key up event
    }
    // and so on...
}
```

Read and handle all incoming frame events.
Populates $event.

Event handling

sdl.h

```
int SDL_PollEvent(SDL_Event *event);

void SDL_Delay(Uint32 ms);
```

Adding new SDL functions to our bindings

sdl.h

What is SDL_Event?

```
int SDL_PollEvent(SDL_Event *event);

void SDL_Delay(Uint32 ms);
```

Adding new SDL functions to our bindings

sdl.h

```
typedef union SDL_Event {
    Uint32 type;
    SDL_KeyboardEvent key;
    SDL_QuitEvent quit;
    // + other members.
} SDL_Event;
```

Defining SDL_Event

When declaring unions, make sure to enumerate all members (variants).

Or at least include the **biggest** member as well as one with the most strict **alignment requirements**.

sdl.h

```
typedef struct SDL_KeyboardEvent {
    Uint32 type;
    Uint32 timestamp;
    Uint32 windowID;
    Uint8 state;
    Uint8 repeat;
    Uint8 padding2;
    Uint8 padding3;
    SDL_Keysym keysym;
} SDL_KeyboardEvent;
```

```
typedef union SDL_Event {
    Uint32 type;
    SDL_KeyboardEvent key;
    SDL_QuitEvent quit;
} SDL_Event;
```

Defining SDL_Event members

sdl.h

```
typedef struct SDL_QuitEvent {
    Uint32 type;
    Uint32 timestamp;
} SDL_QuitEvent;
```

```
typedef union SDL_Event {
    Uint32 type;
    SDL_KeyboardEvent key;
    SDL_QuitEvent quit;
} SDL_Event;
```

Defining SDL_Event members

SDL.php

```php
/** @return ffi_cdata<sdl, union SDL_Event> */
public function newEvent() {
  return $this->sdl->new('union SDL_Event');
}
```

Union objects can be created with the same new() method.

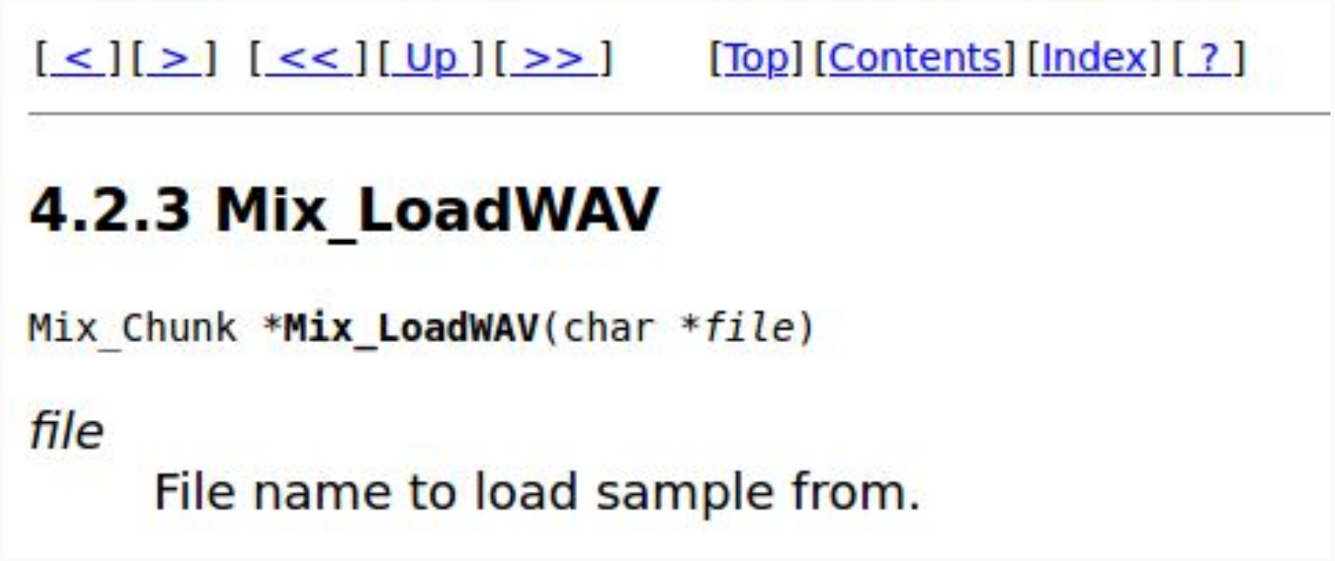
Creating union objects

KPHP game with SDL

Part 3: add SFX & music

OK, Google

How to load WAV with SDL?



[<] [>]   [<<] [Up] [>>]   [Top] [Contents] [Index] [?]

4.2.3 Mix_LoadWAV

```
Mix_Chunk *Mix_LoadWAV(char *file)
```

file
File name to load sample from.

Opening WAV files

sdl_mixer.h

```
typedef struct Mix_Chunk Mix_Chunk;

Mix_Chunk *Mix_LoadWAV(char *file);
```

Loading WAV files with Mix_LoadWAV

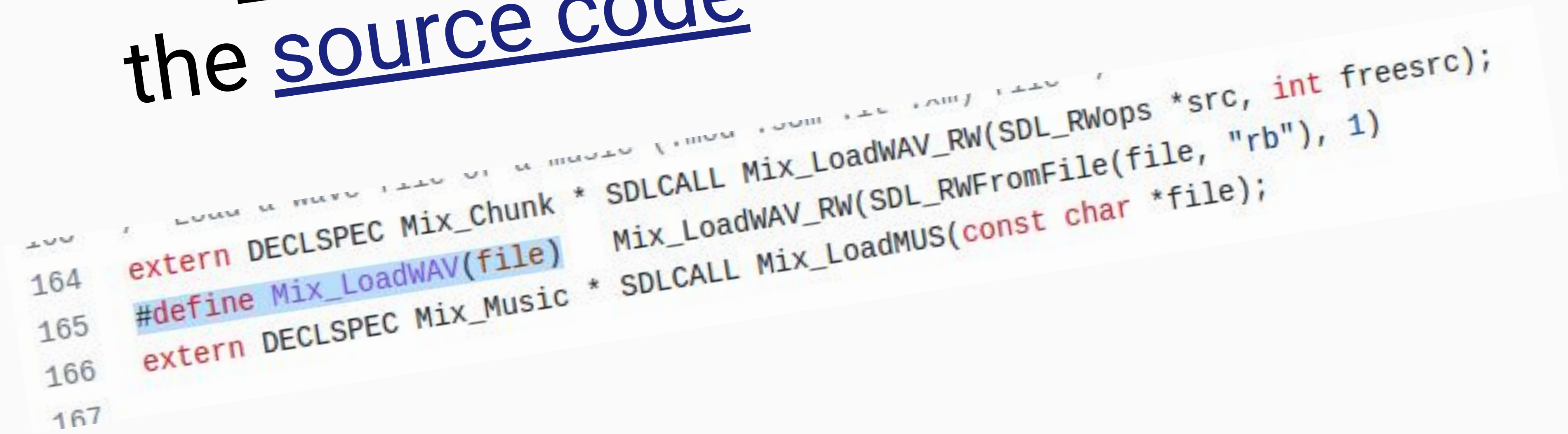
```
$ make game
$ ./bin/game
```

Running the game

```
$ make game
$ ./bin/game
PHP Warning: sdl_mixer library doesn't export
Mix_LoadWAV symbol.
```

Running the game

Let's open the source code



```
extern DECLSPEC Mix_Chunk * SDLCALL Mix_LoadWAV_RW(SDL_RWops *src, int freesrc);
#define Mix_LoadWAV(file)   Mix_LoadWAV_RW(SDL_RWFromFile(file, "rb"), 1)
extern DECLSPEC Mix_Music * SDLCALL Mix_LoadMUS(const char *file);
```

It's a macro, not a function!

```
~~Mix_Chunk *Mix_LoadWAV(char *file);~~

SDL_RWops *SDL_RWFromFile(
  const char *file,
  const char *mode);

Mix_Chunk *Mix_LoadWAV_RW(
  SDL_RWops *src,
  int freesrc);
```

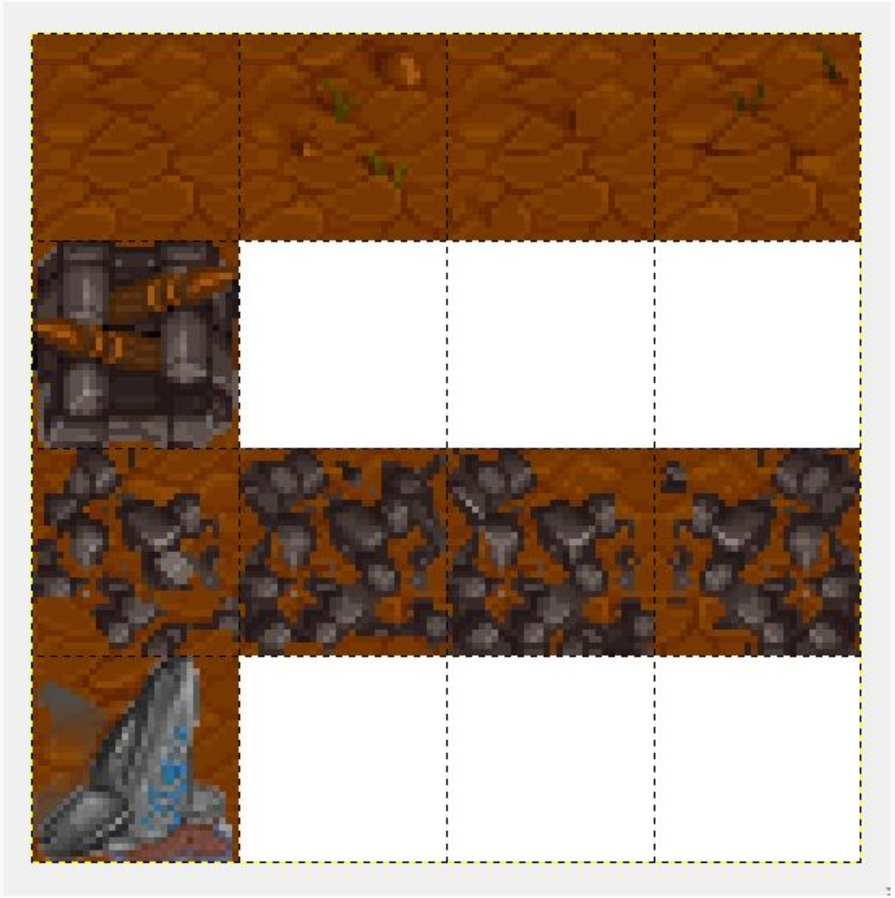
sdl_mixer.h

Loading WAV files

KPHP game with SDL

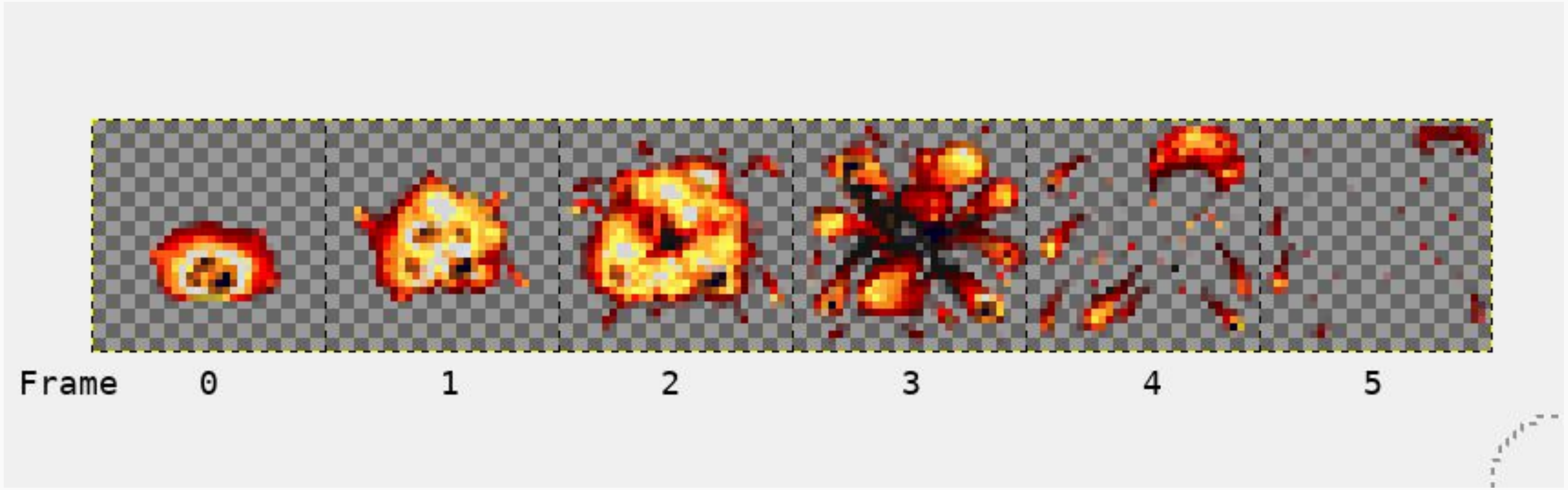
Part 4: other things...

(I can’t cover everything in this talk.)



Atlas textures

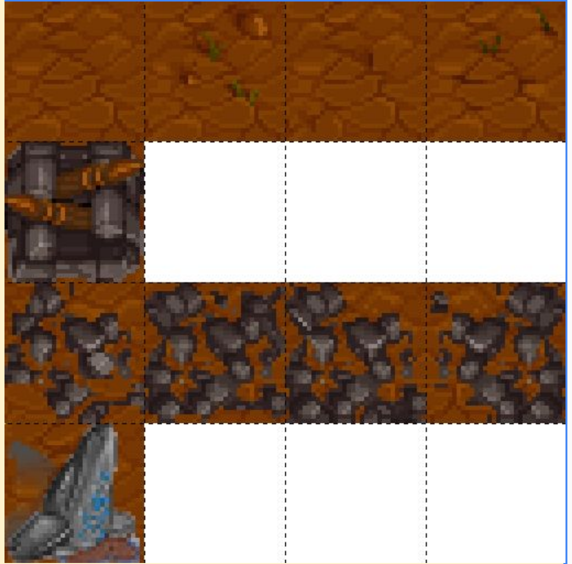
Tiles



Atlas textures

```php
$texture_pos = $sdl->newRect();
$texture_pos->w = 32;
$texture_pos->h = 32;
$texture_pos->x = 0;
$texture_pos->y = 32 * 3;

$sdl->renderCopy(
  $texture,
  \FFI::addr($texture_pos),
  \FFI::addr($pos));
```

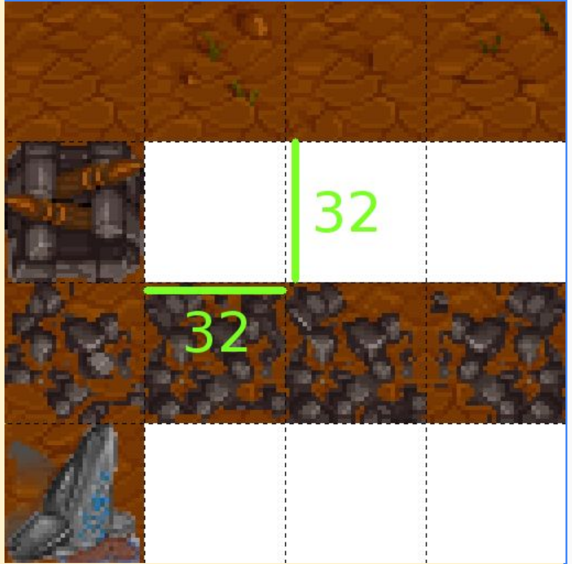
test.php



Atlas texture rendering

```php
$texture_pos = $sdl->newRect();
$texture_pos->w = 32;
$texture_pos->h = 32;
$texture_pos->x = 0;
$texture_pos->y = 32 * 3;

$sdl->renderCopy(
  $texture,
  \FFI::addr($texture_pos),
  \FFI::addr($pos));
```

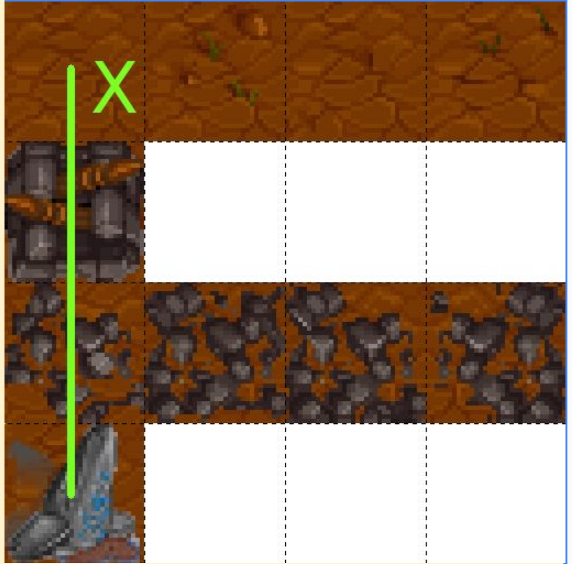
test.php



Atlas texture rendering

```php
$texture_pos = $sdl->newRect();
$texture_pos->w = 32;
$texture_pos->h = 32;
$texture_pos->x = 0;
$texture_pos->y = 32 * 3;

$sdl->renderCopy(
  $texture,
  \FFI::addr($texture_pos),
  \FFI::addr($pos));
```

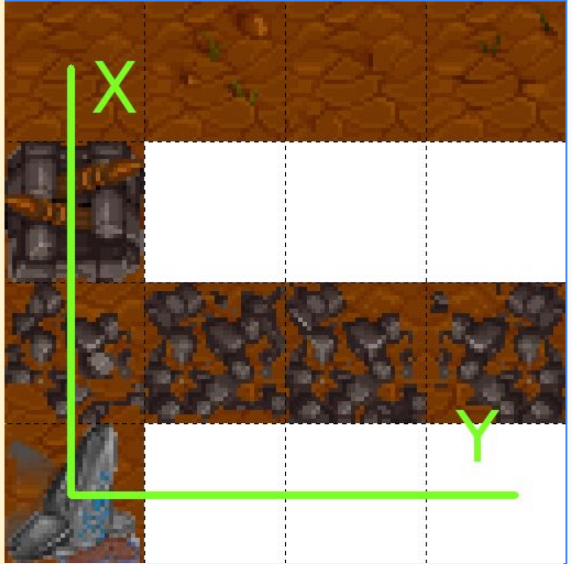
test.php



Atlas texture rendering

```php
$texture_pos = $sdl->newRect();
$texture_pos->w = 32;
$texture_pos->h = 32;
$texture_pos->x = 0;
$texture_pos->y = 32 * 3;

$sdl->renderCopy(
  $texture,
  \FFI::addr($texture_pos),
  \FFI::addr($pos));
```

test.php



Atlas texture rendering

Color.php

```
class Color {
  public int $r;
  public int $g;
  public int $b;
  public int $a;
}
```

Wrapper classes

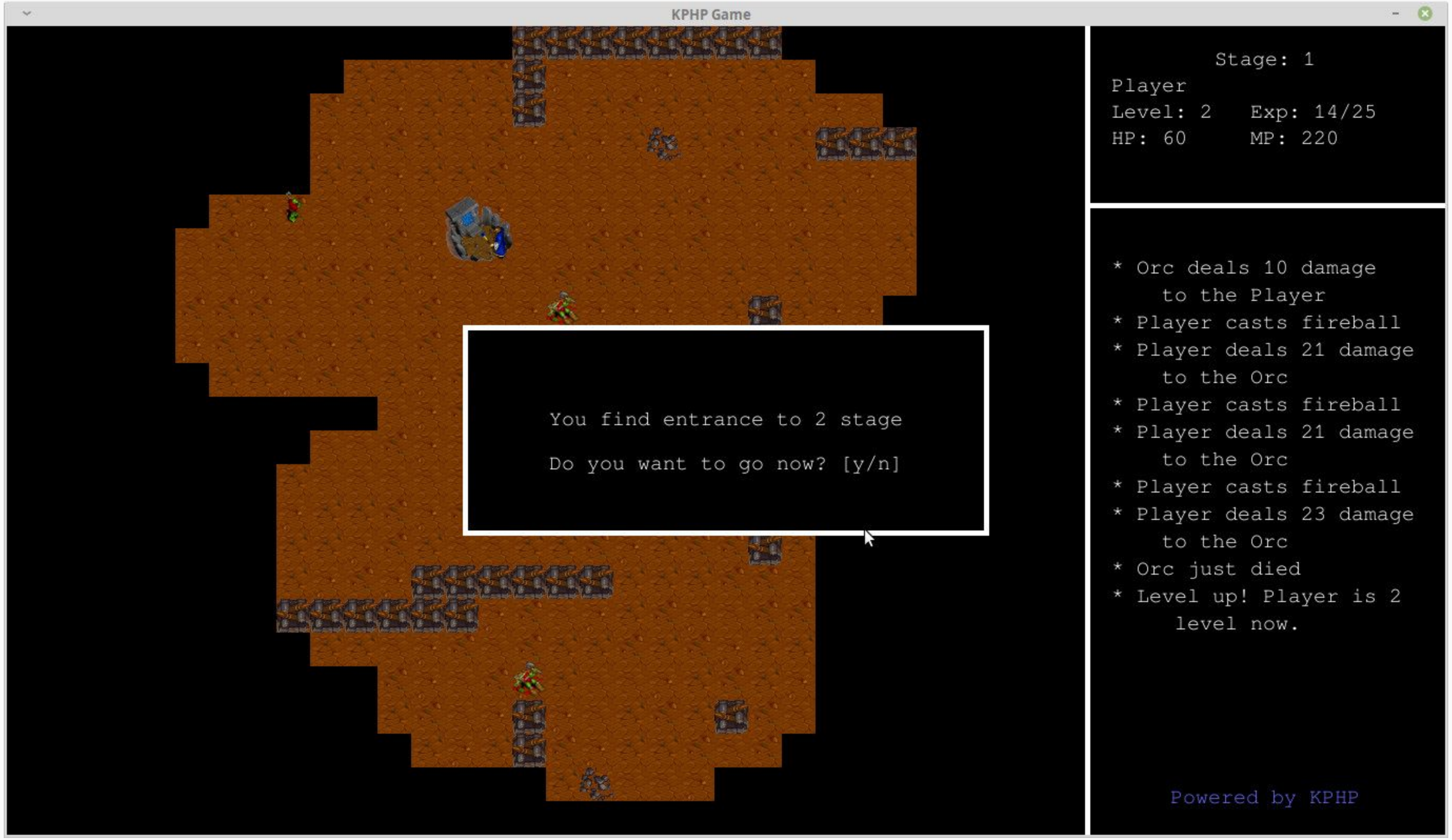
Making the code more readable

```php
/**
 * @param ffi_cdata<sdl, struct SDL_Renderer*> $r
 */
function setDrawColor($r, Color $color): bool {
  $result = $this->sdl->SDL_SetRenderDrawColor(
    $renderer,
    $color->r, $color->g, $color->b, $color->a);
  return $result === 0;
}
```

Making the code more readable

KPHP game with SDL

Part 5: enjoy the result



```
Stage: 1
Player
Level: 2   Exp: 14/25
HP: 60     MP: 220
```

```
You find entrance to 2 stage

Do you want to go now? [y/n]
```

```
* Orc deals 10 damage
   to the Player
* Player casts fireball
* Player deals 21 damage
   to the Orc
* Player casts fireball
* Player deals 21 damage
   to the Orc
* Player casts fireball
* Player deals 23 damage
   to the Orc
* Orc just died
* Level up! Player is 2
    level now.
```

Remember the PHP-KPHP compatibility?

You can actually run that game in PHP too!

KPHP game links

- Game source code
- SDL2 bindings composer package
- KPHP FFI documentation
- Gameplay video

How to use cross-lib types?

How to use cross-lib types?

a.h

```c
typedef struct Foo;
```

b.h

```c
typedef struct Foo;

struct Bar {
  struct Foo *foo;
}
```

Cross-library types

a.h

```c
typedef struct Foo;
```

b.h

```c
typedef struct Foo;

struct Bar {
  struct Foo *foo;
}
```

Incompatible types 'struct Foo*' and 'struct Foo*'

Cross-library types

a.h

```c
void *new_foo();
```

b.h

```c
struct Bar {
  void *foo;
}
```

Use void* and give up on types

Cross-library types: solution 1

test.php

```php
$foo = $a->new('struct Foo');
$a_ptr = FFI::addr($foo);
$b_ptr = $b->cast('struct Foo*', $a_ptr);
```

Use FFI::cast();

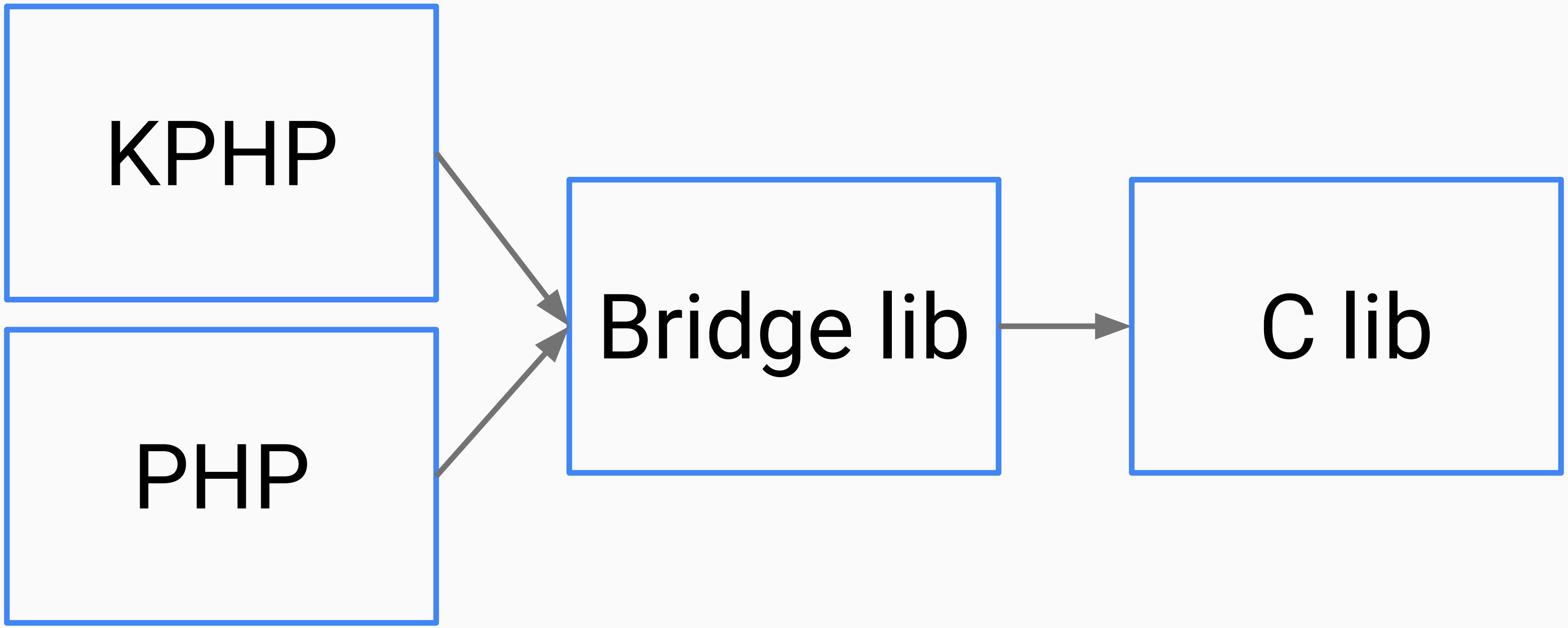
Note: using FFI::cast as instance method!

Cross-library types: solution 2

Is there a workaround to KPHP FFI limitations?

Is there a workaround to KPHP FFI limitations?

Consider using a thin C bridge lib.



Bridge lib contains a glue code and simplified API of a target C lib

Using a bridge lib approach

1. Identify the original C lib API problems
2. Come up with a simpler API that is suitable for FFI
3. Use original C lib in your bridge lib
4. Use bridge lib via FFI in your PHP code

Can I… call Rust from KPHP?

Can I… call Rust from KPHP?

You sure can.

lib.rs

```rust
#[no_mangle]
pub extern "C" fn rust_hello() {
  println!("hello from Rust!");
}
```

Defining a simple Rust function

```
# name = "ffi_lib"
# crate-type = ["cdylib"]

$ cargo build

# library is located at
# target/${build}/lib${name}.so
```

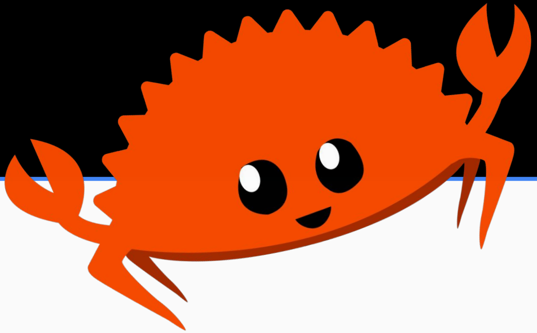
Building Rust project as C shared library

test.php

```php
<?php

$lib = FFI::cdef('
  void rust_hello();
', __DIR__ . '/target/debug/libffi_lib.so');

$lib->rust_hello();
```

Defining a simple Rust function

```
$ kphp --enable-ffi --mode cli ./test.php
$ ./kphp_out/cli
hello from Rust!
```



Building and running KPHP application

KPHP community

KPHP FFI

Extending KPHP using foreign function interface API